Data Fusion at Scale in Astronomy

by

Matthias Lee

A dissertation submitted to The Johns Hopkins University in conformity with the requirements for the degree of Doctor of Philosophy.

Baltimore, Maryland

August, 2017

Abstract

We have arrived in an era where we face a deluge of data streaming in from countless sources and across virtually all disciplines; This holds especially true for data intensive sciences such as astronomy where upcoming surveys such as the LSST are expected to collect tens of terabytes per night, upwards of 100 Petabytes in 10 years. The challenge is keeping up with these data rates and extracting meaningful information from them. We present a number of methods for combining and distilling vast astronomy datasets using GPUs. In particular we focus on cross-matching catalogs containing close to 0.5 Billion sources, optimally combining multi-epoch imagery and computationally extracting color from monochrome telescope images.

Primary Reader: Tamás Budavári

Secondary Readers: Alex Szalay and Randal Burns

Acknowledgments

I would like to express deep appreciation and sincere gratitude to my advisor Prof. Tamás Budavári, for his continuous support, great advice and extraordinary wisdom. His guidance has been incredibly helpful and motivating throughout all of my research. I would also like to express my gratitude to Prof. Alex Szalay, for his strong encouragement to come to Johns Hopkins University as well as his continuous support through out my time here. Thank you.

Dedication

This thesis is dedicated to my parents, who always answered my countless questions and encouraged me to discover and learn, and to my wife, Jessica, who has encouraged me to continue and persist, for putting up with my many late nights and early mornings working towards this accomplishment.

Prior Publications

This thesis is in part derived from prior publications, specifically:

- Chapter 2 is largely derived a poster presented at the 2013 GPU Technology Conference in San Jose, CA.
- Chapter 4 is largely derived from an article[63] published in the Journal of Astronomy and Computing.
- Chapter 5 is largely derived from an article submitted and accepted, but to be published in the Journal of Astronomy and Computing.
- Chapter 7 is largely derived from an article submitted to the Astrophysical Journal.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

We have arrived in an era where we are faced with a deluge of data streaming in from countless sources and across virtually all disciplines; the challenges of managing, processing and correlating these vast volumes of data have become some of the most difficult aspects of science, engineering and business. Over the past few decades, Astronomy and Genomics have pioneered *big data science*, with projects such as the Sloan Digital Sky Survey and the Human Genome Project, pushing the boundaries of traditional science, by driving development of techniques and technologies capable of handling such massive datasets. The majority of enabling advances have come from outside of their specific disciplines, relying heavily on advances in computer science, computer systems and mathematics. Much of modern science would be unthinkable without computer science. Future projects such as the Large Synoptic Survey Telescope (LSST) continue to stretch todays methods far beyond the current state of the

art. LSST is expected to see incoming data acquisition rates in excess of 20 terabyte per night.

As part of this thesis we will explore multiple difficult, large-scale processing problems, and propose new methods and approaches to alleviate these challenges.

In previous decades many projects have relied on the steady advances of increased computing power along Moore's law, often building systems to collect data before useful systems to analyze these had come to market. As of recently we have reached the end of this trend. CPU clock speeds no longer increase, transistor shrinkage has slowed down [Intel tik-tok drop] as we have started approaching the limitations of physics, both in size and power density.

While we can no longer easily make single chips faster, the trend has shifted towards more parallel and distributed architectures. Instead of scaling monoliths up, we now scale out, meaning more parallelism at all levels. The evolution of this is very apparent in industry where *big data* problems need to be addressed at unfathomable scale; cloud deployments, microservices, NoSQL and distributed databases, as well as offloading onto dedicated accelerators such as GPUs and ASICs. The evolution of the GPU has not only pushed the envelope on peak performance and massive parallelism, but also on power efficiency and performance per watt.

1.1 The Rise of the GPUs

Starting in the mid-90s, NVIDIA and AMD/ATI started developing these massively parallel architectures for the computer graphics and gaming industry, which has been continuously pushing for ever higher performance to render the latest video games and simulations. As the capabilities of these cards grew, especially with the advent of programmable shaders, the scientific community quickly realized that these coprocessors could be leveraged by translating their scientific codes to use graphics primitives. In 2006, NVIDIA released the first version of CUDA, which first allowed developers full programmatic access to their graphics cards. This kicked off the era of high-performance computing on Graphics Processing Units (GPUs).

GPUs differ from a standard CPU in many ways, perhaps most drastically in the layout and grouping of essential components. In a traditional CPU, each arithmetic logic unit has it's own control unit, providing maximum flexibility for the executing thread. On a GPU, sets of *streaming processors* (cores) are grouped together into units called a *warp*, where each warp has a single control unit. Sharing a single control unit between many cores lowers the power consumption and the required die space, since, due to their complexity, control units require large amounts of each. At the execution level, this means a single instruction is executed across all cores within the same warp. This is known as Single-Instruction Multiple-Threads (SIMT).[4]

In addition to these architectural differences, GPUs also feature vastly higher bandwidth memory. Standard CPU RAM, such as DDR4 SDRAM tops out at

maximum of 19.2GB/s, whereas GPU RAM, such as GDDR5/GDDR5X top out at 330GB/s. This allows GPUs to read and write data up to 15x faster.

Even though GPUs can feature thousands of core, they are generally clocked slower than CPUs. This plays a major role in the power efficiency, given that doubling the frequency roughly equates to squaring the power consumption of a processor. GPUs tend to run at clock rates around 1GHz, where as modern CPU clock rates are in the 2GHz-4GHz range (depending on application). Differences such as these, allow GPUs to achieve vastly higher numbers of floating point operations per watt. Dong et al[5] investigated this difference empirically, concluding the NVIDIA K20 GPU produced approximately 3x more useful double precision floating point operations per watt when compared to the Intel E5-2670.

As can be expected GPUs also have their caveats. These architectural differences result in a processor capable of extremely high parallel execution, with the requirement that executed software must be much more tailored to the hardware, requiring intricate understanding of hardware details and advanced algorithmic tuning to take full advantage of these features. Due to the often drastic hardware architecture changes between generations, software which ran at peak performance on the current generation may run slower, or at least not at its full potential, on the next generation.

A major caveat is also the overhead of memory transfers, as discussed above, there is an enormous difference in speed between CPU and GPU memory, meaning

that memory transfers between the CPU and GPU are bound by the maximum CPU memory speed. Compounding this problem is the peak throughput of PCIe. Most recent cards feature PCIe v3.0 interface which is limited to approximately 16GB/s. This imposes a transfer-time and latency overhead for every operation needing to move data to or from the GPU. Other factors complicating GPU memory accesses are various different levels of memory, registers, shared, texture and global memory, each having their own advantages and disadvantages, as well as the need to explicitly invoke memory transfers for peak performance. While newer generations of CUDA have the notion of *Unified Memory*, which automatically transfers memory behind the scenes, for peak performance, manual memory transfers and management is still required.

1.2 Scientific Applications of GPUs

The massively parallel architecture, high memory bandwidth, and power efficiency make GPUs great candidates for a wide variety of data-parallel computations. GPUs are specifically well suited for many types of searching, matching, image and signal processing which lie at the core of nearly all computationally-heavy scientific applications. In this thesis we delve into how the combination of power-efficient high-performance hardware accelerators such as GPUs and novel algorithms and approaches allow us to extract meaning out of ever growing datasets. The fusion and

extraction of meaning of large datasets lies at the root of most big scientific, engineering and business challenges. We will examine multiple difficult computer science problems, bridging many scientific disciplines, such as computer science, applied mathematics, statistics, genomics and astronomy. Firstly, we explore the lower limits of power consumption by reducing the idle load of the host system in Chapter 2. Next, in Chapter 3, we turn DNA sequence alignment into a signal processing problem and accelerate it on a GPU using Fourier space cross correlations. In Chapter 4, we discuss a multi-GPU application for cross matching extremely large astronomy catalogs. In Chapter 5, we introduce a fast and robust method for combining and deblurring multi-epoch telescope observations, relatedly in Chapter 6 on scale this multiframe blind deconvolution large datasets, such as the SDSS Stripe 82. Finally in Chapter 7 we take a new spin on the multiframe blind deconvolution and explore its ability to not only combine and deblur images, but also recover sub-band color information from single color observations. All of these accomplishments will be summarized and concluded in Chapter 8.

Chapter 2

CUDA on Low-Power ARM

As science and engineering industries have become more reliant on massive data sets, our need for fast and efficient data processing has grown exponentially. Much of today's new research in fields such as Astronomy, Physics, Chemistry and Genomics rely on large amounts of computation. The demand for super computers such as MARCC's Bluecrab, NCSA's Blue Waters and ORNL's Titan have made this need for computation very evident. The advent of the modern, power-efficient and general-purpose GPU, has seen their introduction into many data processing pipelines as well as modern supercomputers, providing a large gain in performance at a smaller power-consumption premium.

In general GPUs can deliver many times higher Floating Point Operations per Second (FLOPS) per Watt than CPUs can.[5] This has been a great advantage for power strapped data-centers looking to deliver more performance on the same power

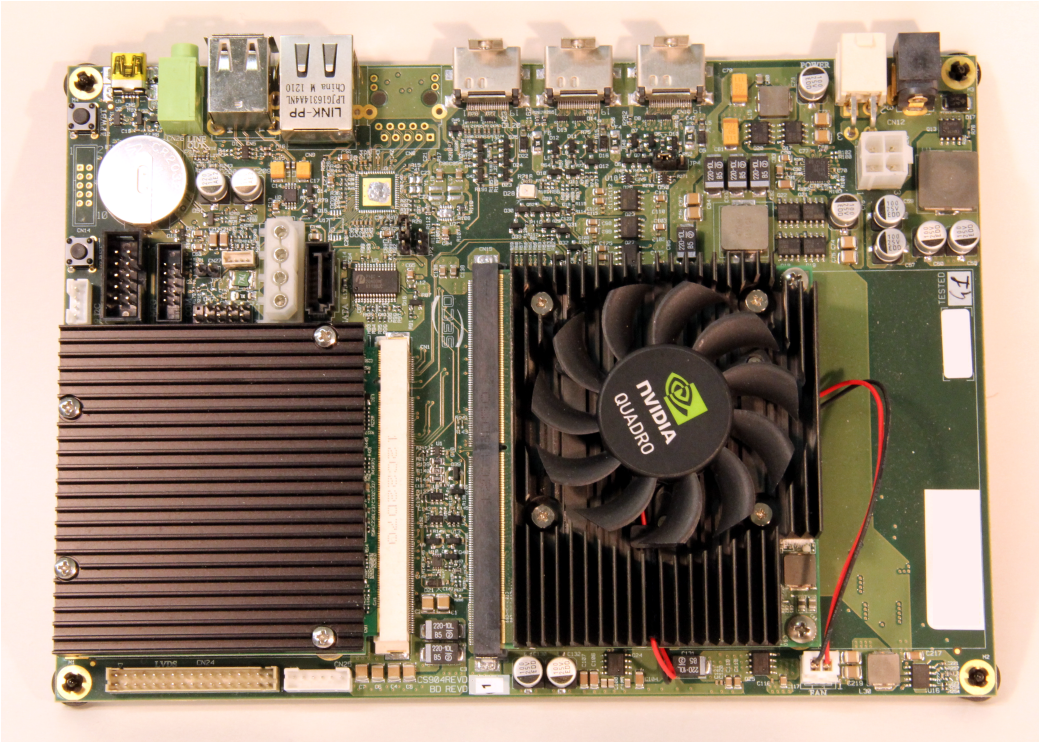

Figure 2.1: SECO CARMA DevKit, Quad-Core Tegra3 CPU and Quadro 1000M GPU

budget. Power savings in data centers count double, since for every watt saved on by the computing hardware, another watt is saved in cooling and power delivery.[6,7] Given that the power consumption accounts for a large portion of the operating budget of a datacenter, makes power efficient computation an important consideration.

As a byproduct of the recent smartphone and tablet market explosion, many companies, including NVIDIA, Apple, Samsung and Qualcomm, have made heavy investments in advances towards high-performance low-power ARM processors. Recently a flood of diverse ARM-based development boards have been released, the $35 RaspberryPi, the Arndale board, the BeagleBone, the PandaBoard and Dell's Copper,

just to name a few. While all of these are low-power ARM-based System-On-A-Chip computers, very few are serious contenders when in delivering high computational power. SECO's CARMA DevKit, see Figure 2.1, is a front runner in this respect. The CARMA combines a ARM-based NVIDIA Tegra3 processor with a powerful Quadro 1000M GPU, yielding a low-power host for a very capable high performance computation co-processor.

2.1 Previous Work

In 2010 Szalay et al.[8] proposed an alternative approach to the traditional high-power compute clusters found in today's data-centers and supercomputers. Szalay proposed an architecture comprised of a larger number of energy-efficient compute nodes coupled with high-performance solid state disks to optimize the relationship between compute and I/O performance, effectively optimizing *Amdahl's number*. Szalay's approach settled on the Intel Atom330-powered Zotac IONITX-A-U with an on-board NVIDIA ION GPU. Intel's Atom processors and their AMD counterparts provide great energy-efficiency in comparison to their high-end siblings, but x86 based processors pale in comparison to the power efficiency of ARM based processors. This is where SECO's CARMA DevKit comes in. The CARMA is a full compute node, featuring dual GigE ports, a SATA port, a Quad-Core NVIDIA Tegra3 processor and most importantly a CUDA enabled 96-Core NVIDIA Quadro 1000M graphics

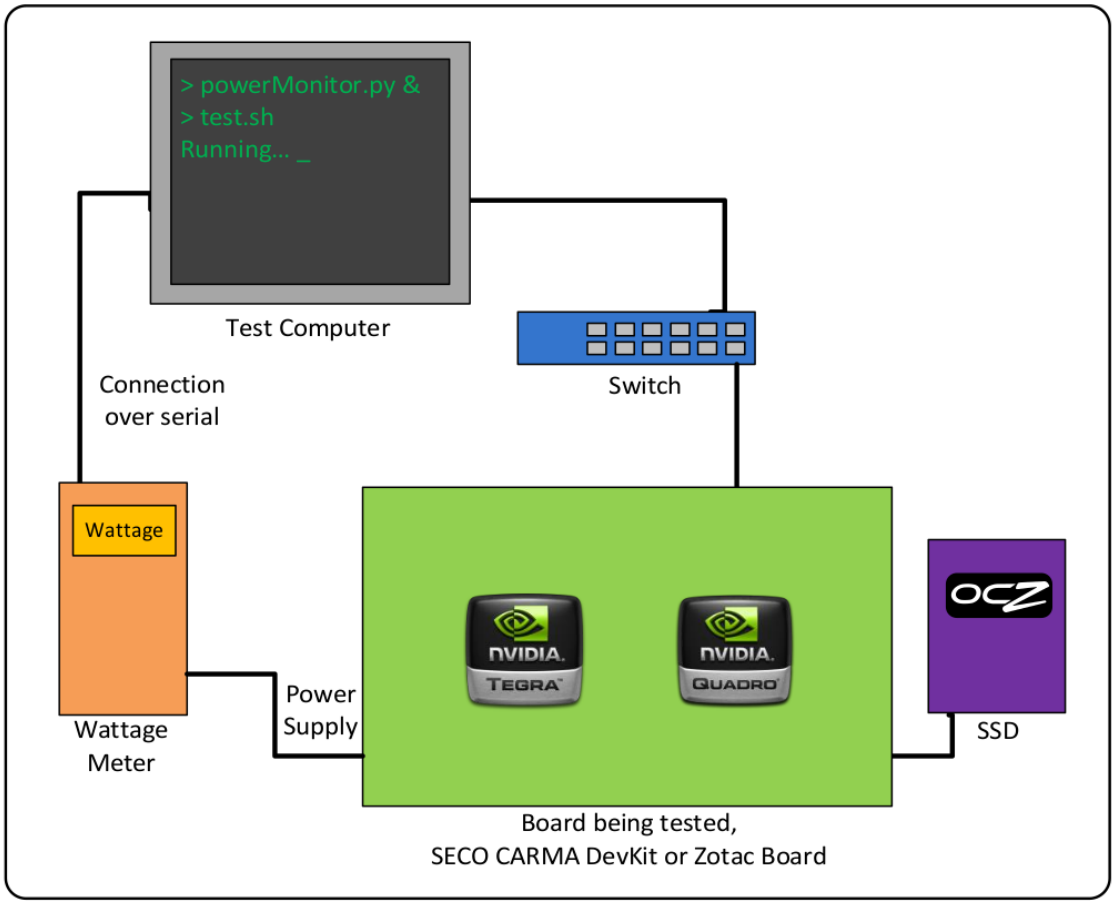

Figure 2.2: Test setup showing all included components.

processor. This is rare set of feature on ARM boards and also the worlds first ARM board to supporting CUDA natively.

Below aim to evaluate the CARMA's performance per watt characteristics and investigate it's use as a candidate for a next generation Amdahl cluster. To test the performance we have setup a power-monitored stress test, featuring a GPU-heavy workload.

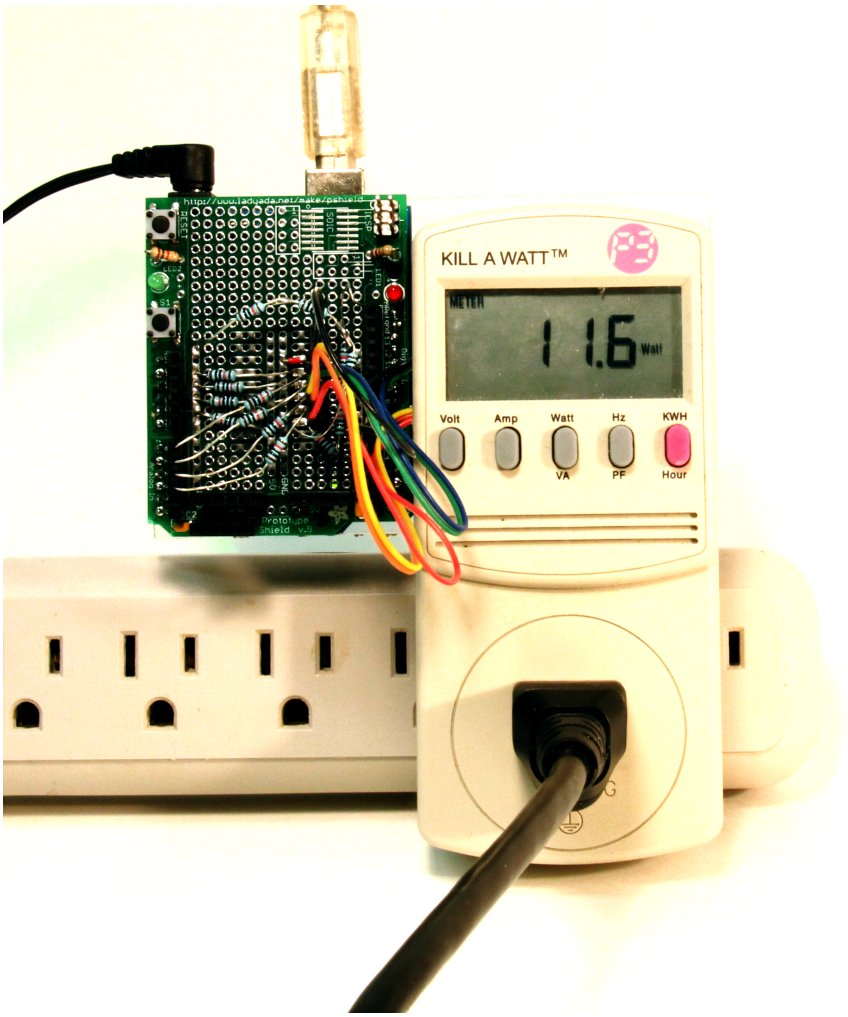

Figure 2.3: Modified Kill-A-Watt connected via FTDI/Serial to test machine.

2.2 Benchmarking Approach

Our goal with this benchmarking is to highlight the combination of an energy-efficient host system for a powerful GPU. Our hardware setup is as follows: the CARMA runs a vanilla Ubuntu 11.04 installation with an ARM-compatible development build of CUDA 4.2 provided by NVIDIA/SECO. Attached to the CARMA is an OCZ Vertex-2 120GB solid state drive containing the test user's home directory. The

comparison candidate, an original Zotac Amdahl node, also uses an OCZ Vertex-2 120GB solid state drive, with a vanilla Ubuntu 11.04 server install configured with a standard CUDA 4.2, see Figure 2.2. In order to eliminate sources of variability in the power consumption, we powered both boards by a fan-less "brick" power-supply. Traditional ATX power supplies consume 5-10 Watts purely for operation of their cooling fan. Both the CARMA and the Zotac board have an on-boad CPU/GPU cooler, we measured both fans to consistently draw approximately one Watt. To adequately stress each of the boards GPU performance, we formulated a simple GPU vector calculation kernel, containing a loop of i iterations with each 16 useful Floating Point Operations. Every GPU thread executes multiple vector operations on one element of each of the two input array, then stores the solution into a result array. To ensure the NVIDIA CUDA Compiler (NVCC) does not optimize out any of the above Floating Point Operations, we examine the generated PTX code, see the annotated PTX in Figure 2.4. Note the 10 highlighted floating point operations. 6x mad (2 FLOPs each), 3x mul (1 FLOPs each), 1x add (1 FLOPs each). The 1x sub at the end is not counted as it is the loop iterator and therefore not a useful operation.

We initialize our input and output arrays with 2^{24} elements and initialize i to 2^{14}. This gives us a grand total of 2^{42} (2^{24} x 2^{14} x 2^4) Floating Point Operations. For power monitoring we use a P3 Kill-A-Watt P4400 which has been modified by soldering 2 sense lines to the output of the P4400's main op-amp. These are then read out over serial through the on-board Analog-Digital Converter(ADC) of an Arduino

```
$Lt_0_3074:
        .loc  14    17    0
        ld.global.f32     %f7, [%rd6+0];
        ld.global.f32     %f8, [%rd8+0];
        .loc  14    9     0
        ld.param.f32      %f5, [__cudaparm__Z5saxpyiiffPfS_S_fff_b1];
        .loc  14    17    0
        mul.f32     %f9, %f7, %f5;
        .loc  14    9     0
        ld.param.f32      %f4, [__cudaparm__Z5saxpyiiffPfS_S_fff_a];
        .loc  14    17    0
        mad.f32     %f10, %f8, %f4, %f9;
        .loc  14    9     0
        ld.param.f32      %f6, [__cudaparm__Z5saxpyiiffPfS_S_fff_fact];
        .loc  14    17    0
        mul.f32     %f11, %f6, %f10;
        mad.f32     %f12, %f8, %f4, %f7;
        mad.f32     %f13, %f6, %f12, %f11;
        .loc  14    9     0
        ld.param.f32      %f3, [__cudaparm__Z5saxpyiiffPfS_S_fff_b2];
        .loc  14    17    0
        mul.f32     %f14, %f7, %f3;
        mad.f32     %f15, %f8, %f4, %f14;
        mad.f32     %f16, %f6, %f15, %f13;
        .loc  14    9     0
        ld.param.f32      %f2, [__cudaparm__Z5saxpyiiffPfS_S_fff_b3];
        .loc  14    17    0
        mad.f32     %f17, %f6, %f2, %f16;
        add.f32     %f1, %f1, %f17;
        st.global.f32     [%rd4+0], %f1;
        .loc  14    18    0
        sub.s32     %r2, %r2, 1;
        mov.u32     %r10, 0;
        setp.ne.s32       %p3, %r2, %r10;
        @%p3 bra    $Lt_0_3074;
$Lt_0_2562:
```

Figure 2.4: PTX assembly code for vector calculations of the Benchmark kernel.

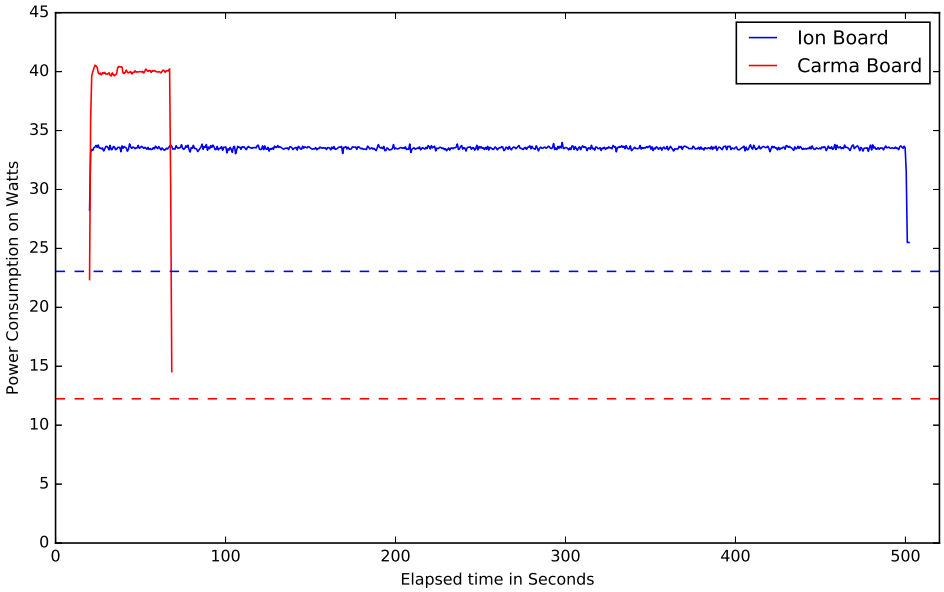

Figure 2.5: Power Consumption of Seco Carma vs Zotac Ion

Duemilanove, see Figure 2.3. Before every test run this device was calibrated using dummy loads to ensure its accuracy. This setup allows for automatic high-frequency (4.4kHz+) readout, monitoring and logging of power consumption. The software for this modification is available under GNU Public License on Github[1]. During a benchmark run, the *test computer* drives the test execution via SSH on the *target board* while also recording power consumption, allowing for fully autonomous testing cycles.

[1]https://github.com/madmaze/serialkaw

2.3 Performance Results

Our test results indicate that the CARMA provides a more than 10x speedup and an 8.65x increase in performance per watt. Out of the box, the CARMA already idles at approximately half (53%) of the Zotac board's idle wattage, see Figure 2.5. There is also a significant difference in the efficiency of the GPUs, the 1000M produces approximately 3.42 GFLOPS/Watt and the ION approximately 0.87 GFLOPS/Watt. It is important to note that these benchmark performance results are based on only one very basic type of workload, meaning we optimized for neither of GPUs. It is apparent that the CARMA outshines the Zotac board in our benchmark, but that is, at least in part, to be expected as we are comparing boards and GPUs of different generations.

All test were repeated 20 times, numbers show in table 2.1 summarize our results. As a measure of quality we calculate a relative standard deviations (RSD), which illustrates the variability during the tests. For the measurement of kernel runtime, we observed a 0.0029% RSD for the CARMA board and 0.0304% RSD for the Zotac board. For the average power consumption, we observed a 5.97% RSD for the CARMA and a 1.125% RSD for the Zotac board.

Overall both of these boards perform well in a low power environment, especially when considering fullsize servers idle at over 100 Watts. Given that compute hardware spends much of it's time sitting idle, this advantage already translates to enormous power savings. In general, given the results of our tests, the CARMA DevKit would

Floating Point Operations per Second:

Board	GPU	Total FLOPs	Time (s)	GFLOPS	Speedup
Carma	1000M	2^{42}	47.6	92.31	10.13x
Zotac	ION	2^{42}	482.8	9.11	1.00x

Total Power Consumption:

Board	Idle Watt	Peak Watt	Ave Watt	GFLOPS/Watt	Efficiency Factor
Carma	12.24 W	40.55 W	39.26 W	2.35	8.65x
Zotac	23.05 W	33.99 W	33.50 W	0.27	1.00x

Power Consumption due to Workload:

Board	Workload Watt	GFLOPS/Watt	Efficiency Factor
Carma	27.02 W	3.42 W	3.92x
Zotac	10.45 W	0.87 W	1.00x

Table 2.1: Power efficiency comparison between Seco Carma and Zotac Boards

provide a significant increase in performance and power efficiency over the Zotac board and would therefore be a viable option for the next generation Amdahl cluster.

Chapter 3

Matching Genomic Sequences on Graphics Processing Units

Much of genomics depends on fast and scalable sequence alignment, whether multiple sequence alignment or alignment of sequence reads to a reference genome. Alignment is a crucially important, but computationally complex problem. Approaches such as Needleman-Wunsch[9] or Smith-Waterman[10] rely on dynamic programming while others use Suffix trees or the Burrows Wheeler Transform.[11,12] An interesting and little studied approach is alignment by cross-correlation. Alignment by cross-correlation yields both regular base-by-base matches as well as complementary matches indicating matching nucleotides from the complementary DNA strand. Outside of the field of genomics, cross-correlation is a well studied method, especially in the field of signal processing where it is often used to determine similarities and

phase differences between signals. These signals are usually in the form of time series, though in our case we will ignore the implied relation to time and just consider it a sequence of intervals.

3.1 Prior Work & Motivation

The idea of using Fourier-space cross-correlation was first entertained by Joseph Felsenstein[13] et al. in 1981. Felsenstein describes a technique which creates four separate indicator arrays of binary values, one for each nucleotide. Each of these arrays is used to encode the occurrence and location of each different nucleotide by marking the location with a 1.0. For example, lets define four indicator arrays, $\{I_a, I_c, I_g, I_t\}$, see Table 3.1.

	A	C	C	G	T	?	A	G	C
I_A	1.0	0.0	0.0	0.0	0.0	0.25	1.0	0.0	0.0
I_C	0.0	1.0	1.0	0.0	0.0	0.25	0.0	0.0	1.0
I_G	0.0	0.0	0.0	1.0	0.0	0.25	0.0	1.0	0.0
I_T	0.0	0.0	0.0	0.0	1.0	0.25	0.0	0.0	0.0

Table 3.1: Translation between DNA sequence, ACCGT?AGC, and Indicator Arrays

When a nucleotide at a given position is unknown we simply distribute our count of 1.0 evenly across all indicator arrays at that position, see position marked with a question mark in Table 3.1. This method can be extended to also reflect more complicated uncertainty values often available for such sequences, we simply distribute the probability across all indicator array, such as $\{0.6, 0.1, 0.1, 0.2\}$ for $\{I_A, I_C, I_G, I_T\}$

respectively. The initial approach outlined by Felsenstein aligns two input sequences, X and Y, by first transcribing them each into their respective indicator arrays and then using four time-domain cross-correlations to find the best alignment between the two sequences. Due to the less-than-favorable $O(n^2)$-complexity, Felsenstein then suggests using a frequency-domain FFT-based cross-correlation which vastly decreases the computational complexity to $O(n \log_2(n))$. Even though Felsenstein's second approach successfully reduces the complexity, it still requires separate cross-correlation for every indicator array. Both Cheever[14] et al. and Rockwood[15] et al. improved on Felsenstein's method by suggesting the use of only one indicator array. Instead of transcribing into multiple arrays of binary values we now transcribe into a single array of complex values, A,T,C,G transcribes to 1,-1,i,-i, see Table 3.2. This reduces the computational effort even more by eliminating multiple FFTs. Rockwood also explored a method of graphing the partial sum of the real parts of the cross-correlation's result to visualize where the similarity between two inputs lies. There are few other

	A	C	C	G	T	?	A	G	C
I	1.0	$-i$	$-i$	i	1.0	0.0	1.0	i	$-i$

Table 3.2: Translation between DNA sequence, ACCGT?AGC, and a single series of complex values

mentions of these method in the genomic literature. Much of the work associated with sequence alignment has focused on alignment of short sequences to much larger sequences. As a cross-correlation requires Fourier-transforms to be performed of the size of the longer sequence, it cannot contend with highly optimized methods available

for short sequence alignment. But for larger sequences, especially for large approximate matches, it becomes more efficient. This method is most efficient both input arrays are of the same size and when that size falls on a power of two.

As the algorithm mostly consists of FFTs and element-wise multiplications, it is an ideal candidate for parallelization, especially as the length of input arrays increase, allowing for more parallelism. The data-parallel nature of this problem is especially well suited for massively parallel processors such as GPUs.

3.2 Cross-Correlation

A time-domain cross-correlation is nothing more than calculating the sum of the element-wise products between two inputs of the same length. Let us call our two input arrays of length n, h and g, and let us refer to this sum as f. f indicates the similarity/correlation between h and g at their default alignment.

$$f(k) = \sum_{j=0}^{n} h(j)\, g(j+k) \tag{3.1}$$

To find their correlation at different alignment, we start shifting g with respect to h such that the overflow at the end of g wraps around to the beginning. Let us call the value of this *shift* k and the sum at that *shift*-position $f(k)$. If we compute this sum for every possible k, then find the maximum $f(k)$, we have found the shift yielding the best alignment, see Equation 3.1. This approach provides a straight

forward alignment indicating which shift maximizes the overlapped between our two sequences.

Unfortunately, this approach grows at the unfavorable complexity of $O(n^2)$. Luckily there is an equivalent algorithm which achieves $O(n \log_2(n))$-efficiency, this is the FFT-based Fourier-space cross-correlation. A Fourier-space cross-correlation achieves it's boost in efficiency by using the Fast-Fourier-Transform,[16] denoted by $\mathcal{F}$ and its inverse $\mathcal{F}^{-1}$. The FFT efficiently transforms it's two input sequences, h and g, from the time-domain into the frequency-domain. Let the $\mathcal{F}(h) = H$ and $\mathcal{F}(g) = G$. A cross-correlation, $\mathcal{C}$, in the frequency-domain is simply the element-wise product, F, of H and the complex conjugate of G. When F is transformed back into the time-domain, we end up with an array of correlations, call it f. Each element in f corresponds to the correlation at a shift equal to that element's index. For example, if $f(6) = 0.5$, the correlation between h and g, when g is shifted by 6 elements, the correlation is 0.5. If we simply find the maximum and its corresponding index in f, we have the shift producing the best correlation, $\mathcal{C}_t$, and therefore the best alignment, see Equations 3.2, 3.3 and 3.4)

$$F = \mathcal{F}(h) \cdot \texttt{conj}(\mathcal{F}(g)) \tag{3.2}$$

$$\mathcal{C} = \max(\mathcal{F}^{-1}(F)) \tag{3.3}$$

$$\mathcal{C}_t = \arg\max(\mathcal{F}^{-1}(F)) \tag{3.4}$$

3.3 Implementation

This tool, gpuFFTMSA[1], is implemented in python, heavily leveraging *NumPy*, *pyCUDA* and *pyFFT*'s CUDA-module. The tool consists of 3 main components: *pyFFTalign.py* which is a wrapper allowing for command-line arguments and controlling the reading and writing of Sequence data. *dataObj.py* is an object containing the sequence data itself and the methods for modifying and managing it. To instantiate a dataObj we pass it the name of the sequence, the source filename and the raw DNA or RNA reads. Upon instantiation, the DNA/RNA is automatically transcribed into a complex indicator array. Additional it exposes functionality for verification of successful transcription, returning of padded versions of the raw and transcribed data. *aligner.py* contains the core functionality for actually computing the alignment. It consists of two main codepaths, a CPU-implementation and a GPU-implementation. Both are essentially identical with the exception of the GPU setup and communication.

For simplicity we will discuss the CPU implementation and then contrast the changes necessary for the GPU acceleration. The main function takes two parameters, each being an array of dataObjs, H and G. We begin by looping over every H_i contained in H and for every H_i we will iterate over every G_j in G. This produces every combinations of H_i and G_j. For every pair, we calculate a FFT-based cross-correlation. As FFTs are most efficient on power of 2 input sizes, we zero-pad our

[1] https://github.com/madmaze/gpuFFTMSA

input accordingly. We then execute compute the FFT cross-correlation, resulting in an array of correlations, f. Note the prominence of the maximum correlation in Figure 3.1.

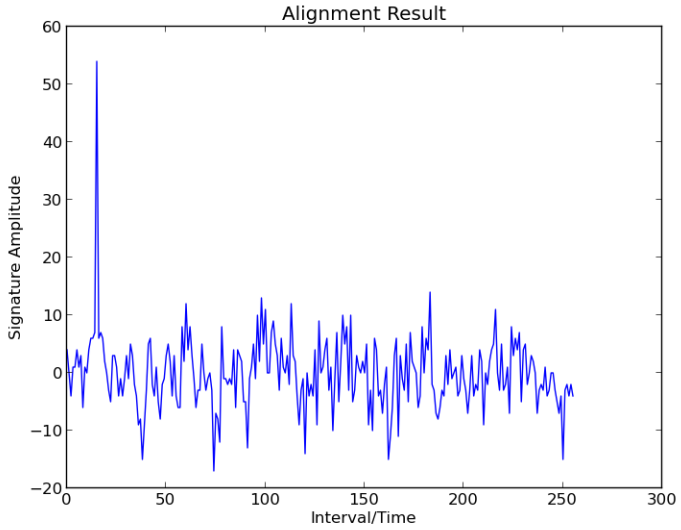

Figure 3.1: Array of correlations, note the strong peak indicating a good alignment at around 19.

The GPU equivalent varies slightly. The looping, padding and math are identical, first difference is that all of these calculations are executed in GPU memory requiring a few extra steps to loading our arrays on to GPU. The only other difference is the that FFT "plan" must be precomputed and preconfigured to a certain size. The code currently waits for the computation to finish before returning from the GPU, this is an optimization to be explored in the future. The rest of the GPU implementation is mostly the same, with the exception of moving data back to CPU memory before we identify the best correlation.

Now that we have discussed the procedure of how we arrive to our result array f

lets have a look and see what information we can gleam out of the results besides what shift provides the best correlation. In the introduction we mentioned that this method can also give us complementary matches, see Figure 3.2, for match resulting from the complement of the sequence used in Figure 3.1. Both of these figures match exactly, but often in genomics we have partial matches. When we have a sequence with read errors, specifically replacements, we will still see a peak at the same locations, but the correlation will be lower. If we have errors such as insertions or deletions, we will get "split peaks", signifying we have matched 2 pieces, but at different shifts, see Figure 3.3 for example.

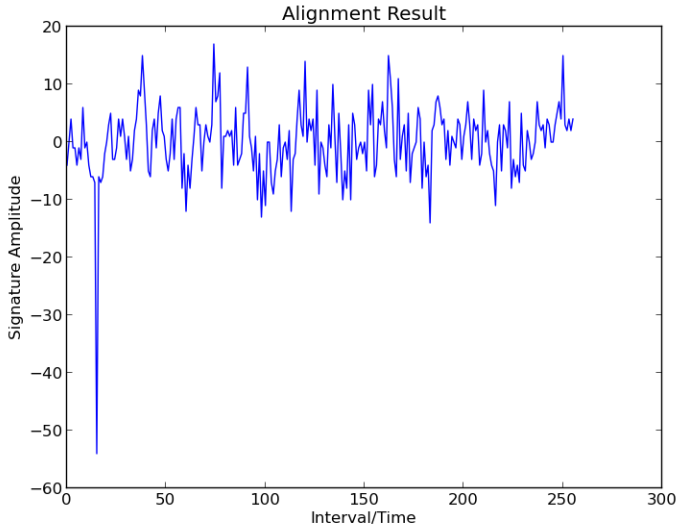

Figure 3.2: Array of correlations, note the strong negative peak indicating a good complementary alignment.

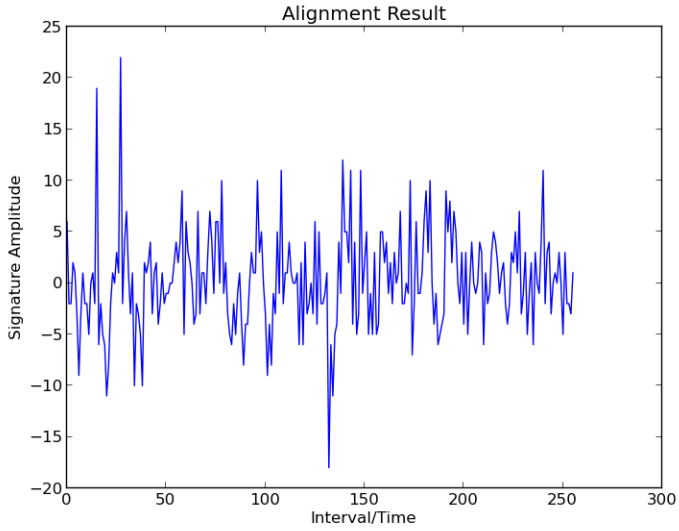

Figure 3.3: Array of correlations, "Split peaks" occur with insertions or deletions are matched.

3.4 Results & Conclusion

The goal of this research is to study whether using GPU acceleration along with this method allows for a reasonable processing rate. CPU implementations of this method are incredibly slow, requiring multiple seconds per alignment, this is realistically too slow. Our bare-bones GPU implementation delivers an over 20x speedup in comparison to the CPU implementation, see Figure 3.4, at which point this method becomes feasibly for research usage. An interesting feature of this algorithm is that, due to the inner workings of the Fourier-space cross correlation, it's performance is tied to the size of the larger sequence. While this might seem like a detriment, this allows for higher efficiencies when the search sequence is extremely long. Due to the recent advances of shotgun sequencing, where reads tend to be very short, typically

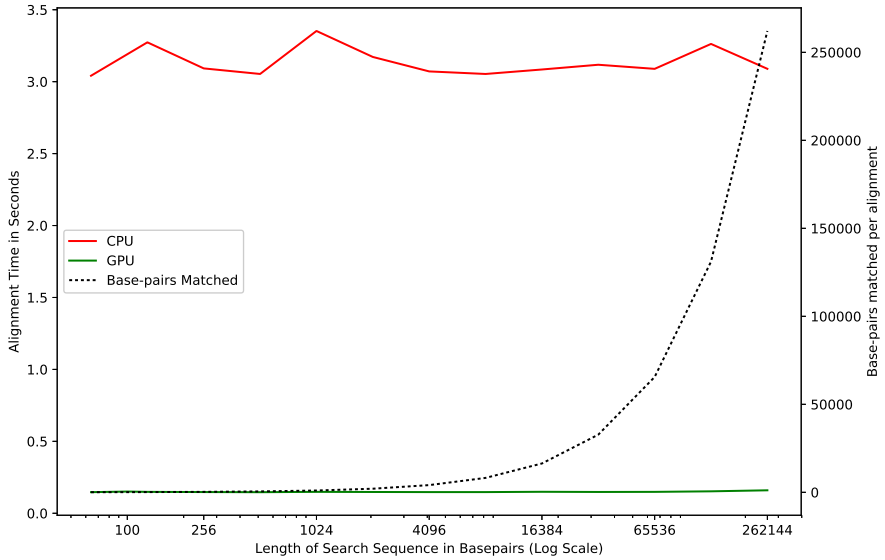

Figure 3.4: Alignment Performance matching against a 5.3 million base-pair reference sequence

less than 500 base pairs in length, most of the research has been focused at aligning short reads against a very long reference genome. The resulting algorithms tend to be very good at aligning short sequences, but inefficient in aligning longer sequences. In Figure 3.4, we see that even though the search sequences grow *6 orders of magnitude*, both the CPU and GPU, exhibit nearly identical performance across these lengths. In the same Figure, we also plot the efficiency, given by the number of base pairs aligned for each

The current GPU implementation, while over 20x faster, is ripe for optimization. Optimizations such as batch processing and subdividing the reference genome into multiple subsegments could likely gain another factor of 5 performance increase.

Also, the use of cuFFT, the NVIDIA FFT library, will likely improve FFT performance. Another avenue for improvement would interleaved FFT executions and memory transfers. Currently, for development, each FFT is executed in series, allowing for time to go wasted on the CPU and GPU while one waits for the other. While the results of this research are already validating this approach, given additional optimizations, this approach can prove valuable in many applications, especially in large scale searching and index operations, where the reference genome could be cached and index after having already been converted into Fourier space, saving an additional 1/3 of the FFTs.

Chapter 4

Faster Catalog Matching on GPUs

Modern telescopes produce vast volumes of data every night. With the current and upcoming advances in technology, survey data-sets are growing at a tremendous pace. To maximize the scientific value of each experiments we often need to combine their observations with other surveys. The identification of objects across multiple catalogs and surveys can lead to new discoveries and breakthroughs but the speed of matching is a limiting factor when combining the large outputs of many experiments. A number of studies have focused on the statistical aspect of this challenge[17–20] but they all rely on fast 2-way matching engines to find candidate associations first.

The current solutions including the SkyQuery[21,22] and the CDS X-Match Service[23] use hierarchical indexes and space-filling curves to accelerate the process.[24,25] While their performance is great they are far from the speed of interactive data exploration, the ability to do on-the-fly catalog federation would be a game-changer. To illus-

trate the big-data aspect of the project, lets consider a relatively small scenario of matching GALEX (50 million objects) and SDSS DR7 (150 million objects), a naïve implementation would require 7.5 quadrillion (10^{15}) comparisons.

We describe a novel approach that takes advantage of the extreme parallelism available on modern GPUs as well as an efficient method for reducing the total number of comparisons. The tool we present can crossmatch at rates of over a trillion candidate pairs per millisecond. We will limit our investigation to matching only two catalogs but without assuming that they are sorted or indexed in any way ahead of time. This choice is motivated by the fact that n-way associations can be built up by 2-way matching using the best guess direction of the partial matches.[17]

4.1 Divide and Conquer

The combinatorial scaling of a naïve matching approach can be remedied by quickly eliminating pairs at large separations. This is often achieved by partitioning the sky and considering only nearby areas instead of the entire sphere. These heuristic algorithms often rely on hierarchical division schemes such as Igloo,[26] Hierarchical Triangular Mesh (HTM),[24] HEALPix[25] or SDSSPix.[27]

4.1.1 Building on The Zones Algorithm

Our approach follows the division scheme of the *Zones Algorithm*[28] which employs a much simpler scheme. It breaks up the sky into constant declination rings called the *zones*; see Figure 4.1. All zones have the same *height*, h, measured in angle. For details on choosing a good value for h, see Section 4.2. A zone identifier is assigned to each source i based on its declination $\delta_i \in [-90°, 90°]$ as given by

$$Z_i = \left\lfloor \frac{\delta_i + 90°}{h} \right\rfloor \ , \tag{4.1}$$

where $\lfloor\ \rfloor$ indicates the floor function rounding down to the closest integer. Sources with the same values are in the same zones, which creates easy and computationally cheap division of sources across catalogs. The simplicity of this equation enables the immediate implementation in any language unlike the more complicated schemes listed above. This also fits well with the indexing facilities of relational database management systems (RDBMS). For example, the SkyQuery solution relies on zones for the largest matching problems, as an optimal query plan can essentially stream through the data on the hard drive with high efficiency and little memory overhead. The key for crossmatching is to use the same zone layout across all catalogs and eliminate all zone pairs that are farther than a specified search radius. The resulting set of zone pairs effectively mitigates any overlap needed to account for positional errors. Within the zones it can be beneficial to further sort by the right-ascension

of the objects to quickly eliminate candidates that are too far apart even before calculating the separation between sources.

4.1.2 Layers of Parallelism

Modern GPUs can run thousands of threads in parallel. In addition they also have superior memory bandwidth as compared to CPUs. The capacity of RAM, however, is somewhat more limited than that of today's servers. With these parameters in mind, we design our architecture to take maximum advantage of multiple GPUs in a single box. We will further assume that one of the catalogs, the smaller, can fit in the computer's memory, which will increase the speed of the execution. This is not, however, a significant constraint considering that 1 billion sources can be stored in 24GB of memory with 8-byte numbers for object identifier and the two celestial coordinates.

We slice the input catalogs into suitable *segments* that fit on the GPUs and build a job scheduler to process pairs of these segments from two catalogs. For the purposes of these next sections, let us consider two unsorted catalogs, `Catalog_A` and `Catalog_B`, each catalog consisting of objects identified by their `ObjId` (int64_t), `RA` (double) and `Dec` (double). Our method breaks down into two main levels of parallelism. At the high level we distribute the problem across GPUs and at the lower level we parallelize across the many-core architecture within each GPU.

4.1.2.1 Preparing the Segments

At runtime we begin the loading process of each catalog by subdividing each into multiple segments of size n. The size is chosen such that two segments, plus the overhead needed for processing, can fit into GPU memory. While GPU memory itself is very fast, transfers to and from are comparatively slow and hence should be kept to a minimum. The division of the catalogs into segments is purely a construct allowing us to manage the data more effectively in order to fit the data onto the GPU and minimize GPU-memory transfer overhead.

We begin by loading the smaller of the two input catalogs, into CPU memory, segmenting it as we go. After each segment is read from disk, it is loaded onto a GPU and sorted by the zone identifiers Z and right-ascension using the C/C++ CUDA *Thrust* library.[29] This is done via a custom comparator implemented as a functor which on-the-fly and in parallel calculates the Z values. Arithmetics on the GPU are very fast and repeated calculations of the zone identifier does not slow down the process as we save on memory transfer, which is the typical bottleneck. Sorting by zone identifiers is only part of the battle, we also need to identify the zone boundaries, enabling us to easily separate out zones at execution time. This has also been implemented in parallel using the `thurst::lower_bound` and `thrust::upper_bound` search functions, which are based on Thrust's vectorized binary search.

We then loop over the larger catalog, reading one segment per available GPU. These segments are also loaded onto the GPUs and sorted by zones. At this point the

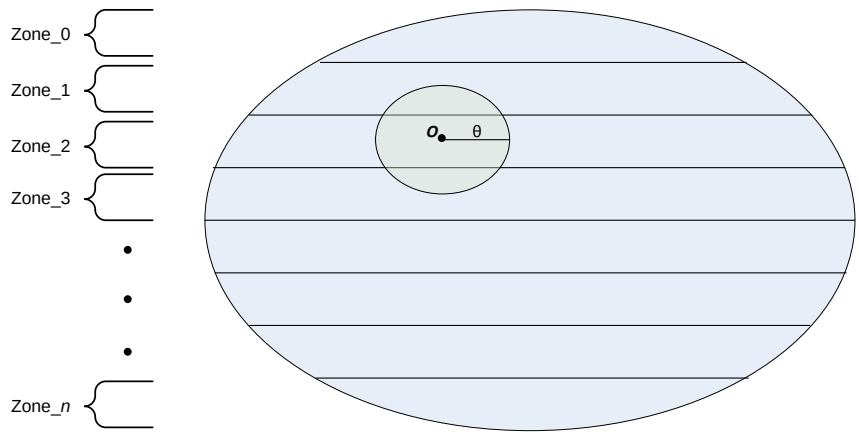

Figure 4.1: The Zones Algorithm: the coordinate plane is subdivided along one axis into small strips of height h called *zones*. To find an object within a search radius, θ, all zones are searched which overlap with search radius.

preparation has taken place and the system is ready to loop through the segments of the catalog loaded in CPU memory.

4.1.2.2 Jobs and Workers

A dynamic execution environment is implemented where a pool of worker threads, one thread per GPU, wait for jobs consisting of two segments, one from each input catalog. A *job manager* keeps track of the current memory content of each GPU and assigns the jobs for processing. This is done via a greedy algorithm that performs close to the ideal in realistic settings for 2-way matching. Each job is a unique pairing of one segment from each catalog, such that every segment of `Catalog_A` is paired once with every segment of `Catalog_B`. The scheduling of jobs is also optimized, such that preference is given to jobs which share a common segment with the previous job,

Input : Catalogs A and B, each containing objects identified by (id, ra, dec) and (id', ra', dec'), respectively

```
// Load Segments for smaller Catalog
```
gpu-dispatch *each Segment of A* **do**
    Load Segment from disk;
    Copy to GPU;
    Sort by Zones and RA;
    Identify Zone boundaries;
    Copy back from GPU;
end

```
// Loop over larger Catalog
```
while *Segments in B remain* **do**
    gpu-dispatch *one Segment of B per available GPU* **do**
        Load Segment from disk;
        Copy to GPU;
        Sort by Zones and RA;
        Identify Zone boundaries;
        Copy back from GPU;
    end
    `// Enqueue one GPU job per possible combination of loaded Segments from` A `and` B
    gpu-dispatch *each Segment-Segment pairing* **do**
        Compute distance metric for every object within each Zone-Zone pairing;
        Accumulate comparison results;
    end
end

Output: List of Object-Object pairings (id,id'), where (ra, dec) and (ra', dec') are considered a match

Algorithm 1: High-level Crossmatch Processing Steps

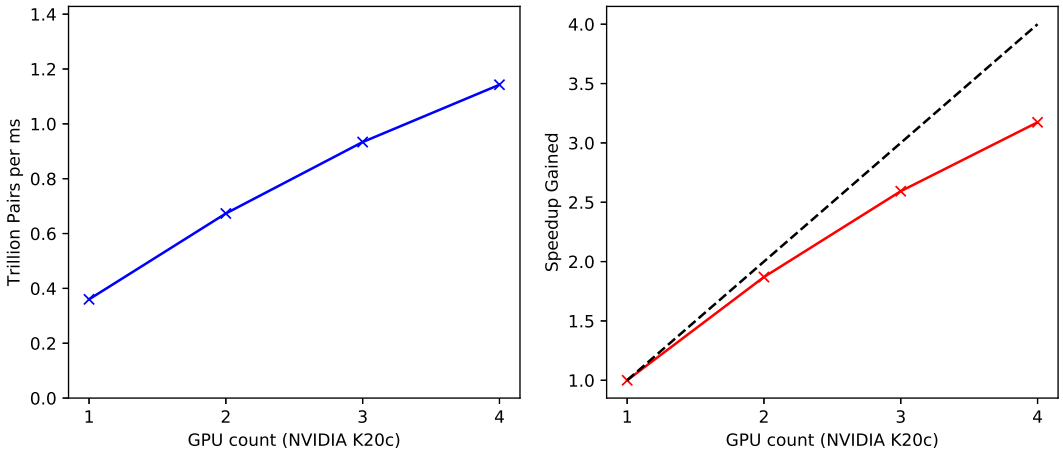

Figure 4.2: Same-task performance characteristics of matching two catalogs of 450 million each, using a zone height of 2 arcseconds and a search radius of 1.5 arcseconds. *left:* Matching rate scaling in trillion candidate pairs per milliseconds based on the number of GPUs. *right:* Speedup gained by scaling up the number of GPUs. The dashed line indicates perfectly linear scaling.

allowing us to avoid unnecessary overhead of reloading the same segment multiple times.

Each worker independently computes the crossmatch between the two segments applying the Zones Algorithm to only consider relevant pairs of zones given the search radius. A custom CUDA kernel (Xmatch) is launched for each pair of zones, which is the heart of the implementation.

4.1.2.3 Matching on the GPU

A typical modern GPU features thousands of cores arranged on multiple Streaming Multiprocessors (SM). For example, the NVIDIA Tesla K20c device has 13 SMs with 2496 cores in total, which can run 26624 threads (2048 threads per SM) in parallel

at any given time. To map sets of GPU threads across the available SMs, threads are arranged in a hierarchy of `blocks` and `grids`, where a `blocks` can be thought of as an array of threads and `grids` are an arrangement of `blocks`. Both can be 1-, 2-, or 3-dimensional arrays as appropriate for the applications. The notion of a `block` is important as the threads within are guaranteed to run on the same SM and hence can communicate very efficiently. It is important to note that each core has access to different types of memory with vast differences in access speeds. For example, shared memory – directly part of the SM, – can be accessed in only 11-22 clock cycles, where as global memory takes 200-800 clock cycles (access cycles dependent on hardware generation). The SM automatically attempts to hide much of the global memory access latency by swapping out warps which are currently waiting for memory transfers to return. Additionally global memory accesses should be coalesced, meaning memory will perform best when all threads in a warp access a contiguous block of memory. This allows the GPU to perform a single memory transaction, instead of separately fetching memory for each thread. More in-depth information on these topics can be found in the CUDA Guide.[30]

Our implementation takes these optimizations into account wherever possible. We launch kernels such that every source-source distance calculation is assigned to a separate but suitably arranged GPU-thread allowing for coalesced memory accesses for parallel read and write operations, which are often the capital expense in many-core applications. Indeed, finding the nearby pairs in the sorted `zones` is very fast

when run on thousands of threads in parallel. The challenge is to efficiently save the results. The custom `xmatch` kernel computes all the pairwise distances and flags the candidate associations as a boolean output into a shared memory array. Naturally, this is very sparse, making it impractical for saving. To optimize memory utilization and therefore to minimize the overhead of extra data transfers, we use a series of parallel algorithms within the `block` to find matched candidates. The core of these algorithms are a parallel prefix scan[31] and bisection algorithms that convert to a sparse representation within shared memory where only the indices of the candidate associations are saved. At the end of this procedure the same threads that previously found the candidates, are used to make a coalesced write to copy the results to global memory for ultimately returning to CPU memory. We repeat these steps for each zone-zone comparison, after which all matches are copied back from the GPU and written out to results files.

4.1.3 Available from C++ and Python

Our code is available publicly under the MIT open source license on Github[1]. We welcome any feedback and pull requests. The codebase mainly consists of a `Command line tool` and a `Python interface`. The `Command line tool` is implemented in C++/CUDA, reading a simple binary format directly available from most databases: simply `ObjID`, `RA` and `Dec` columns, 8 bytes each, totaling 24 bytes per object. We

[1] http://matthiaslee.github.io/Xmatch[32]

also provide a small python utility to convert CSV files to and from this format. The minimal argument set consists of 3 required arguments, `catalog_A`, `catalog_B` and `segmentSize`, all other arguments have reasonable defaults. The optimal segment size should be set such that two segments plus about 15% overhead can fit into GPU memory. The full command-line options are shown in Usage 4.1. The `Python interface` provides direct access to the C++/CUDA-native function using Python's ctypes interface. The exposed interface offers an identical function-set to the command line tool. For example, the Python lines

```
>>> from Xmatch import crossmatch
>>> crossmatch('catalogA.idradec',
               'catalogB.idradec',
               segSize=100000000,
               radius=1.5,
               zoneHeight=1.5,
               numGPU=6)
```

would work with the binary catalog files to produce the matches as expected.

4.2 Evaluating Performance

Our tool can compare two catalogs of 150M objects in approx. 1.5 minutes and two catalogs of 450M objects in approx. 9 minutes on a single NVIDIA K20c GPU. As we scale up the number of GPUs, we see much improved performance. With four NVIDIA K20c GPUs we reduce the runtime for the 150M×150M case to 45 seconds and the 450M×450M case to under 3 minutes, achieving a crossmatch rate of over 1 trillion candidate pairs per millisecond; see Figure 4.2.

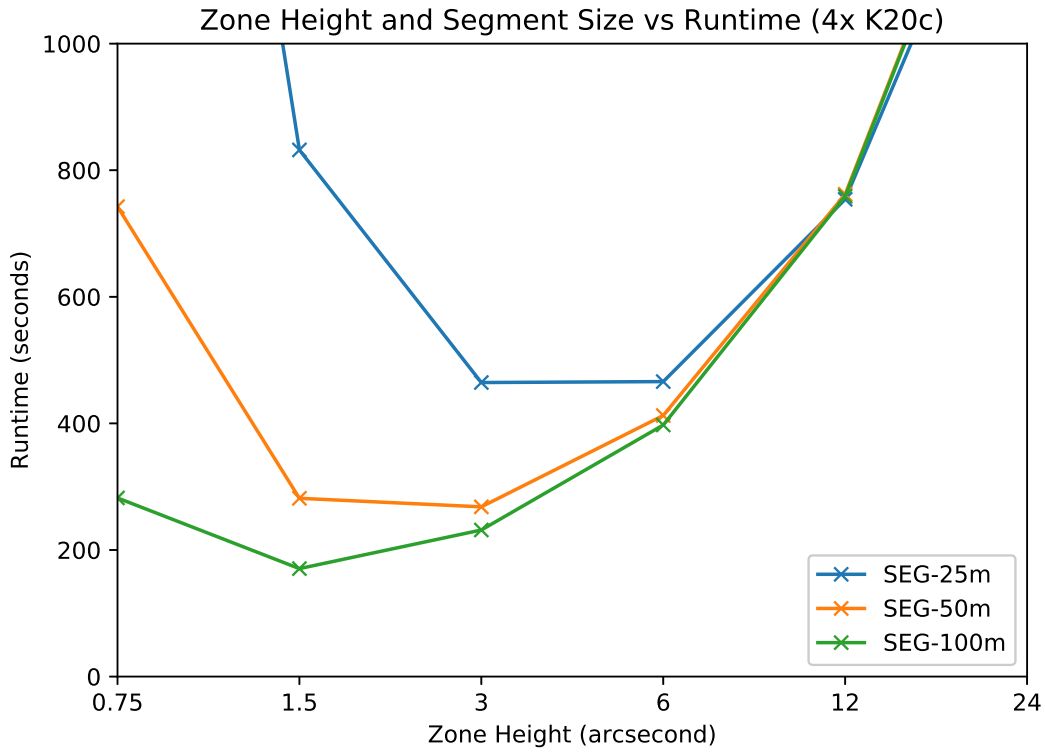

Figure 4.3: Runtime Performance as a function of the zone height and segment size. As the segment size increases to the maximum possible on an NVIDIA K20c GPU (100 million), we approach best case performance. The smaller the number of total segments, the more time can be dedicated to the actual distance computations, therefore increasing the performance. Also note the effect of zone height, depending on the search radius, the zone height dictates how many matches are computed per zone-zone comparison. When the zone height is too large, we loose the advantage of limiting our search area and when the zone height is too small the overhead of launching extra kernels overtakes the advantage of limiting our search area.

```
Usage: ./BoostXmatch.x [options] file(s)
Options:
-o [ --outdir ] [=arg(=out)] Output directory path
-m [ --maxout ] arg (=0)     Maximum output per job, default to (jobsize)^2
-r [ --radius ] arg (=1.5)   Search radius in arcsec, default is 1.5"
-z [ --zoneheight ] arg (=0) Zone height in arcsec, defaults to radius
-t [ --threads ] arg (=0)    Number of threads, defaults to # of GPUs
-n [ --numobj ] arg          Number of objects per segment
-v [ --verbose ] [=arg(=9)]  Enable verbosity (implicit level 9=ALL)
-x [ --overwrite ]           Overwrite output directory
-h [ --help ]                Print help message
```

Usage 4.1: Help output for the BoostXmatch tool.

Our method has 3 main variables which control how the matching algorithm is run, each of which also effect the overall runtime. We have the h value, which controls the height of each zone (arcseconds), the θ value, which controls the search radius (arcseconds) and n, which controls the segment size. For our testing we set the search radius, to 1.5 arcseconds, which exceeds the realistic maximum angular object separation for SDSS,.[17] The search radius obviously has a large impact on our search time, the larger the search radius the more objects need to be compared. The search radius should be large enough to account for measurement errors but not larger than necessary.

In order to achieve the fastest and most efficient matching given available hardware resources, we want to maximize our Segment size, which is limited by the total memory available on each GPU. For our tests on the K20c this number is approximately 100 million objects per segment. Finally we fix the zone height, which dictates how many sources will be compared for each zone-zone matching. For small values

of h, the majority of the time is spent launching CUDA kernels, but large values of z increase the number of sources in each zone, forcing us to do more comparisons than necessary. Our testing concluded the optimal value for 4 GPUs is 100 million objects per segment and $h = 1.5$, for a two factor comparison of the effect of adjusting segment size and zone height, see Figure 4.3.

4.3 Conclusions

In this study we take a fresh look at efficient catalog matching, one of the biggest challenges of modern observational astronomy in the multi-wavelength and time-domain era. Building on ideas of the Zones Algorithm, we developed a novel approach to finding associations on the many-core general-purpose graphics processors in today's video cards. Our C++ and CUDA implementation achieves unprecedented speeds and can process two catalogs of 450 million entries each in less than 3 minutes using four NVIDIA Tesla K20c cards. Smaller catalogs and subsets of interest can be processed in interactive times with these new tools which means that on-the-fly data fusion in services such as SkyQuery can scale to the next generation of time-domain surveys.

An easy to use Python interface as well as a Python helper script being tested currently and available on Github along side the current C++/CUDA implementation. Future works includes tuning of the thread layout in accordance to newer compute

capabilities to better utilize latest GPU architectures, dynamic zone sizing to minimize zones with low numbers of comparisons as well as the integration of this engine into the SkyQuery cluster solution.

Chapter 5

Robust Image Deconvolution

In the new era of astronomy surveys, dedicated telescopes observe the sky every night to strategically map the celestial sources. The next-generation surveys are capable of such high speed that repeated observations become possible, opening a new window for research to systematically study changes over time. The key requirement for time-domain astronomy is the development of sophisticated algorithms that can maximize the information we gain from the data. The image processing approaches we present here are motivated by these astronomical challenges but are not specific to such exposures and are expected to work for long-range photography regardless of the content of the images.

Algorithmically deblurring single telescope images has had a long history. At first during the 1970s, when Richardson[33] and Lucy[34] independently introduced a Poisson-based iterative deconvolution method, generally known as the Richardson-Lucy, and

then during the early days of the Hubble Space Telescope when researchers rushed to remove the blur induced by the misaligned optics.[35–39] Most of these techniques rely on a variation of an iterative deconvolution using a *known* point spread function (PSF). The main difficulty with such methods is preventing numerical degeneracies caused by the presence of noise and poorly constrained parameters.[40] Even more challenging is when the PSF is unknown. Some techniques, such as Fish's the iterative non-negative matrix factorization (NNM) approach,[41] have shown limited success simultaneously solving for the PSF and the latent image. The aforementioned approaches are limited by the total amount of information contained in the input image and the PSF (if available).

Ground-based imaging, astronomy or long-range photography a like, suffers from varying distortions introduced by turbulence in Earth's atmosphere. Multiple exposures, however, provide an opportunity to break the degeneracy of the varying PSFs and the constant background image. Blind multiframe deconvolution uses multiple images to simultaneously solve for the latent image and the individual PSFs. This blind deconvolution method is based on Fish's approach of iteratively solving for both the PSF and the latent image, the difference being that new observations are regularly introduced in order to add more information into the deconvolution, this is the basis of the approach described Harmeling.[42]

The current state-of-the-art techniques in astronomy for extracting and combining repeated observations are relatively simple methods when compared to other areas

of image processing. This is in part due to the incredibly high dynamic range of the images, as well as the occurrences of large noise dominated area which tend to handicap methods from other areas. The two most common techniques used are *lucky imaging*[43] and the *coadding* of observed images.[2,38] Both of these use linear combinations of the input images, which enables proper noise propagation, although the full co-variance matrix is rarely produced.

Lucky imaging is the process of only choosing the observations with the very best PSF and throwing away the rest, often over 90% of the original data is discarded. This yields sharper images, but with a low signal-to-noise ratio. A *Coadd* image is the combination of multiple images, produced by generating a pixel-by-pixel mean or median image, therefore suppressing noise and outliers. However, considering that the input images have different PSFs, one needs to first convolve them to match the worst acceptable blur before combining the pixel values. This results in a high signal-to-noise ratio but also in an overall blurrier solution than the majority of input images. In practice, usually a combination of both of these methods are applied; only the best images are chosen and then co-added, meaning that a large amount of potentially useful data is discarded in the process.

In a perfect world we would like to clean up every single observation without throwing away any useful data, and restore each of them to pristine condition with infinite resolution, but this is impossible. It is, however, possible to extract and combine information across all observed frames and produce a few sharper images of

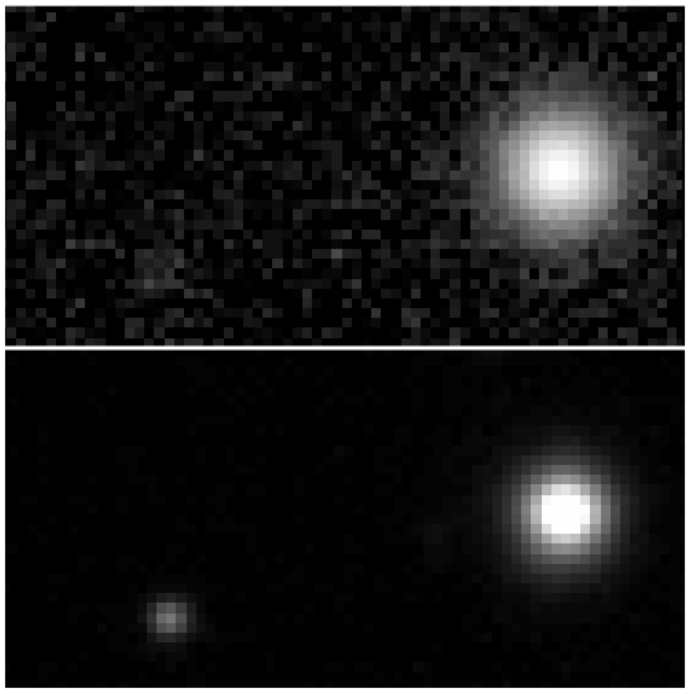

Figure 5.1: Shown above is an example of Sloan's Stripe 82 observations, displaying the problematic features found in these images. *Top* shows an observation with low signal-to-noise and a large PSF, *Bottom* same section of the SDSS Coadd,[2] showing improvement in signal-to-noise and definition of sources over the plain observation.

higher resolution, exposing sources and features previously hidden in blur and noise. True reconstruction is computationally complex and therefore slow and laborious. In order to keep up with the growing volume of data, we need fast, statistically sound tools to explore and deblur these images in near real time.

We demonstrate the application of our method on the Sloan Digital Sky Survey (SDSS),[44] which over a span of 7 years imaged a large part of the southern hemisphere with roughly 80 fold coverage. This area known as *Stripe 82*[45] is an ideal testbed for demonstrating the power of our new methodology. The overall quality of Stripe 82 images varies widely, see Fig. (5.1). Some images are blurry, some are faint and

noisy, but most are a combination of those. Annis et al.[2] produced a state-of-the-art coadd of this region, which we use as a reference in our experiments. Due to the extremely high dynamic-range we encourage the reader to view the included images on a computer screen, as print materials have difficulty reproducing the full range of contrast.

We present a novel approach for extracting information from atmospherically distorted repeated observations in order to produce a deblurred, low-noise, higher resolution image. In section 5.1, we discuss previous work as well as the general approach. In section 5.2 we introduce our robust improvements. In section 5.3 we discuss the application, results and performance of our method and finally, in 5.4 with conclude with future work and a summary of our achievements.

5.1 Deconvolution as Likelihood Optimization

Before we explore removing the blur, we must formally understand how the blur is induced into our observations. For the mathematical description we consider one-dimensional images represented by (column) vectors but all derivations and equations are similar and apply in higher dimensions as well. In particular here we will focus on 2-D images.

Our model is relatively simple: Each observation y_t at time t, is modeled as the

convolution of the underlying *true* image, x and the PSF f_t, on top of which there is measurement noise ϵ_t,

$$y_t = f_t * x + \epsilon_t \ . \tag{5.1}$$

Our goal is to fit this model to all observations in time $\{y_t\}$. Going forward we will make the differentiation between the underlying *true* image, x, and its estimate, $\tilde{x}$, likewise we will distinguish between the observation, y_t, and our reconstruction, $\tilde{y}_t$. Considering that a convolution is a linear operation, we can represent the 1D convolution with f by a matrix F, such that $Fx = f * x$, this also holds true for the equivalent 2D convolution. Hereafter we use this convention to representing matrices with capital letters and vectors with lower case symbols.

The literature features a variety of methods for deblurring with a known point-spread function (PSF). Especially common are methods which maximize the Poisson likelihood, such as the Richardson-Lucy[33,34,46] deconvolution or the more noise resilient damped-Richardson-Lucy.[47] There are also blind methods, e.g., Fish,[41] which do not require a *known* PSF, but instead solve for it as part of the iteration. This is a crucial feature as in most applications, PSFs are not inherently known.

While the noise in CCD observations follows a Poisson distribution, cf. the number of electrons in CCDs proportional to the number of photons in each pixel, its applicability is often limited because creating a complete model for an image is not practical due to many known and unknown contributions (gradients from moon, thin clouds, etc.) The image processing pipelines correct and calibrate the input images.

Estimates of the sky background are subtracted and while noise estimates for each pixel remain accessible, the transformed pixel values will have different noise properties. Fortunately the background counts in typical images are high enough that a Gaussian likelihood is a good approximation to the likelihood function. The multi-frame blind deconvolution method by Harmeling[42, 48] uses a quadratic cost function, which corresponds to the Gaussian limit. The resulting formula is similar to that of the Richardson-Lucy. We can solve for the model image, $\tilde{x}$, which minimizes the residuals in all pixels,

$$\tilde{x} = \arg\min_{x} \sum_{\text{pixels}} [y - Fx]^2 \ . \tag{5.2}$$

In order to solve for the $\tilde{x}$-image update we use an adaptation of Harmeling's multiplicative update formula

$$\tilde{x}_{t+1} = \tilde{x}_t \odot u_t \tag{5.3}$$

with

$$u_t = \frac{F^T y_t}{F^T F \tilde{x}_t} \tag{5.4}$$

where $\odot$ symbol and the fraction indicate element wise multiplication and division. We consider the fraction to be the update image, u_t, derived from the observation y_t, which defines how adjustments are introduced into the model image. This update image is then applied to our model, as shown in 5.3.

There are multiple ways to solve for the PSF, Harmeling[42] proposes using the LBFGS-B Algorithm,[49] Homrighausen proposes a "Fourier deconvolution" method[40]

and Fish[41] proposes simply using the same update formula for the estimation of the PSF as for updating the model image. The latter is possible since the convolution, $f * x$, is commutative, meaning that if we can use an update formula to solve for the model, we must also be able to rewrite it for the PSF. To do so, we simply interchange the occurrences of the PSF and the model in our formula. Essentially we alternate holding the f and the $\tilde{x}$ constant while estimating and updating the other. We assume the PSF to be constant across our images, but these methods could be extended to use multiple separate PSF or use methods such as Lauer's approach to spatially-variant PSFs.[50]

To regulate our solutions, we constrain both $\tilde{x}$ and f to be non-negative and initialize them to non-zero values. While it is possible to initialize $\tilde{x}$ to a constant value, we found initializing with the average of a few observations speeds up convergence and yields better final results with fewer processed observations.

It is important to prevent erroneous zero values from cropping up in our model and PSF, as once an element becomes truly zero, no future multiplicative update will be able to change it. This is especially tricky as our intended background, for our model image, is as close to zero as possible without exactly reaching it. Floating point underflow can be part of this problem and therefore needs to be dealt with appropriately.

In practice a number of problems occur due to random noise in the data and numerical degeneracies. In the following sections we describe our novel methods for

overcoming these limitations.

5.2 Improving Stability and Convergence

5.2.1 Robust Statistics

Arriving at a good estimate for the PSF image is crucial for successful deconvolution. With the unmodified update formula, the cost function that is being minimized is an L_2 norm, therefore featuring a squared term. Extreme outliers in the residual, especially early on in the estimation before the PSF has been well formed, can overtake the residual, forcing convergence in those areas before the rest of the image. This often leads to PSFs more resembling of a Dirac-δ than a realistic PSF.

To curb the impact of those very large values, we borrow elements of robust statistics and modify our quadratic cost function to be an M-estimator with a robust ρ-function.[51] The solution is still iterative in nature and hence lends itself well to our method. We adjust our cost function with the ρ-function,

$$\tilde{x} = \arg\min_x \sum_i \rho\left(\frac{r_i(x)}{\hat{\sigma}_i}\right) . \tag{5.5}$$

In practice the solution to this general problem of robust statistics is obtained by

minimizing a weighted L_2 problem

$$\tilde{x} = \arg\min_x \sum_i w_i \, r_i^2(x) \ , \tag{5.6}$$

where the weight of each pixel w is computed for its residual based on the current solution of plugging $\tilde{x}$ into $\mathbb{W}(t) = \rho'(t)/t$ function,[51]

$$w_i = \frac{1}{\tilde{\sigma}_i^2} \, \mathbb{W}\left(\frac{r_i(\tilde{x})}{\tilde{\sigma}_i}\right) \tag{5.7}$$

In particular we apply the *bisquare* function, also known as the *Tukey bi-weight*,

$$\mathbb{W}(t) = t\left[1 - \left(\frac{t}{k}\right)^2\right]^2 \mathbb{I}(|t| \leq k) \, , \tag{5.8}$$

which corresponds to a ρ -function that is quadratic for low values but approaches a constant for large arguments, essentially limiting the contribution of the largest residuals in the cost function. Figure 5.2 illustrates the difference between the terms in our robust cost function. For more details we refer the interested reader to the discussion of the *bisquare scale* in.[51]

The threshold at which the ρ-function deviates from being quadratic, is tuned by scaling the residuals, which therefore controls where the dampening starts. In order to relate this tuning parameter to the quality of the image, it is natural to define this scaling in units of σ. This controlled dampening of the residuals greatly increases the

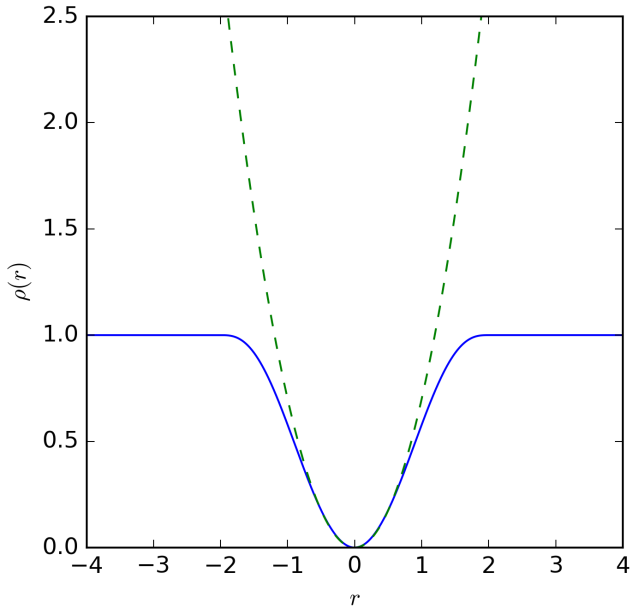

Figure 5.2: *ρ-function* associated with the *bisquare* family of functions. Note the dotted line indicates the contribution of a purely quadratic function.

quality of the estimated PSFs and therefore also the resulting image. More detailed results are presented in section 5.3.4.

5.2.2 Convergence

The addition of the weighting as described in the previous section helps constrain the PSF estimation and therefore the areas around objects. On the other hand, the noise-dominated areas *between* objects remain under-constrained, resulting in the occurrence of background artifacts.

The cause of these artifacts had been a long standing problem for this method, until we noticed that as the deconvolution approaches convergence, the useful and

reasonable updates to our model image become smaller, more sporadic and clustered around sources. Conversely, the updates in the regions between sources, where values are already low and close to zero, can become extreme due to noise present in the observed image. For example, if a background pixel in our model is near zero, but noise in the observed image wants to push the value higher, update factors of 1000x or more are not uncommon. In most cases the resulting pixel value will still be very small and will fluctuate around zero, cancel each other out. Unfortunately in some scenarios these persist as artifacts, ie. bright speckles across the noise dominated areas of the image.

To prevent these updates from affecting the model image we introduce an update clipping function,

$$u'_t = \max\left\{1/d,\, \min(d,\, u_t)\right\} \tag{5.9}$$

where $d > 1$, which limits the maximum impact an update is allowed to have on any single pixel. The closer the parameter d is to 1, the higher the impact and therefore the more conservative the updates will be. When d is large, the clipping has virtually no impact. This approach vastly cuts down the number of background artifacts.

Limiting updates in such a way has no real drawback on the dynamic range in practice. For $d{=}2$ the contrast becomes 2^{40} with only 20 iterations.

In fig. 5.3 we show an update image clipped with $d = 2.0$ (*right*) and the corresponding current model estimate (*left*). Observe how the gray areas, where the update is between 0.5 and 2.0, fall directly around the regions where objects are

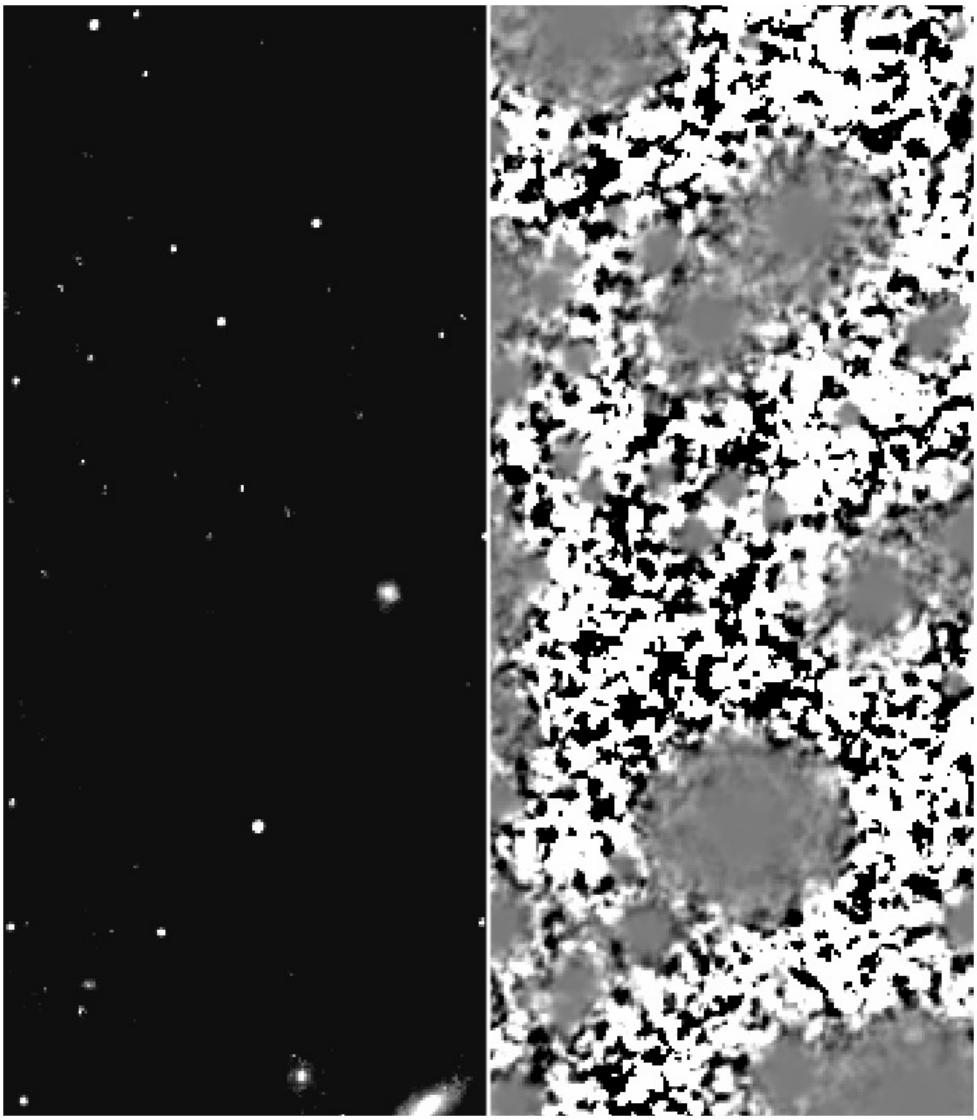

Figure 5.3: A typical update (*right*) to our image model (*left*) contains the most reasonable updates clustered around objects. The values between objects, the background, are already near zero, so any noise or non-uniform background subtraction can produce extreme and unreasonable update values. Note: the coloring of the update image is such that the areas that fall below the bottom clipping are colored black and the areas that are above our clipping range are colored white. The gray areas are values which we consider to be reasonable.

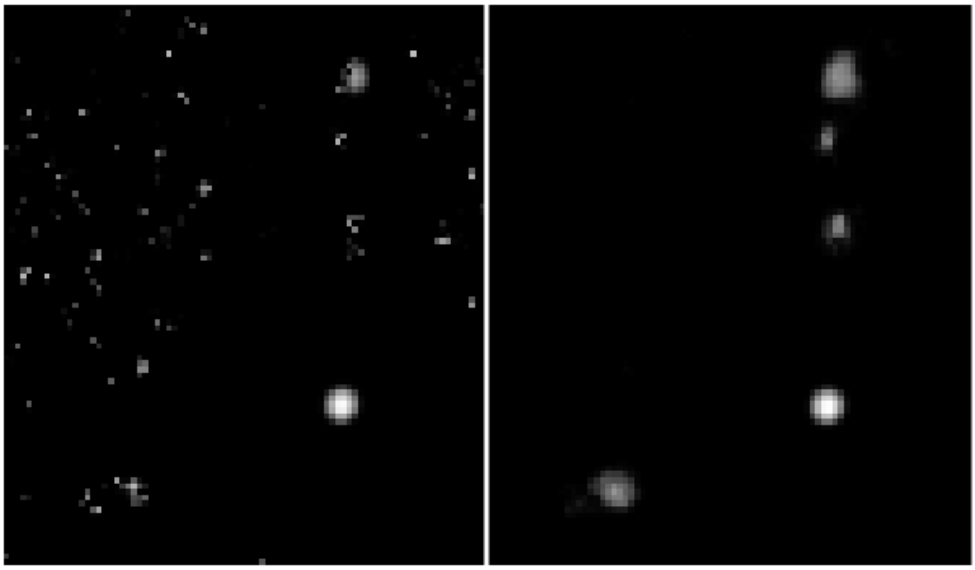

Figure 5.4: Updates to our model image fluctuate most in areas between sources, without dampening these updates, speckles (*left*) get introduced into the noise-dominated background and faint sources can get disrupted by extreme erroneous updates. By limiting the absolute magnitude of these updates we get much more coherent result (*right*).

located in the current estimate. The areas that either appear pure white or black are where the update was originally above 2.0 or below 0.5 and therefore have been clipped respectively, meaning we only allow each pixel to be either doubled or halved with any update. The resulting effect of this clipping can be seen in fig. 5.4, the left image shows some typical artifacts, a speckled background in noise dominated areas, as well as the adverse effects of extreme erroneous updates to faint objects. On the right, is the result of an identical deconvolution with the Update Clipping enabled.

5.3 Application

In the following sections we describe the more traditional components of our method, as well as the overall algorithm. We present the results of our method when applied to real data and also quickly discuss our technical implementation and it's performance.

5.3.1 Pixel Censoring

Our linear model cannot represent omnipresent artifacts such as saturated pixels, cosmic rays, etc. Masking these pixels allows us to process real images containing these artifacts. We introduce binary *mask* images that zero out the known-bad areas and leave the remainder untouched. Each incoming observation y_t, has its own mask associated with it, which for our tests had been pregenerated using masking data available from SDSS. Formally the mask application can be written as a matrix multiplication with a diagonal matrix W, which enters the update formula as follows,

$$u_t = \frac{F^T W y_t}{F^T W F \tilde{x}_t} \ . \tag{5.10}$$

Relatedly flux from objects located just beyond the edge of the image as well as the abrupt edge of zero padding used with FFTs can cause artifacts along the mask's edges, which over multiple iterations can creep into the PSF and begin corrupting the model. To prevent these artifacts, we taper the masks towards zero around the edges

and any masked out object, therefore smoothing out these artificial hard edges.

5.3.2 Super-Resolution

Access to multiple images of the same objects, also gives us the ability to increase the spatial resolution beyond that of the source images. Based on the nature of our images, we can expect each to have a slightly different relative sub-pixel shift, meaning we can extract and combine that information into a higher resolution image model. To do so, we introduce a *downsampling* operator B that lowers the resolution of an image. If the model image $\tilde{x}$ has a factor of 2 higher resolution than the observations y, then B would halve the resolution of $\tilde{x}$ in order to make it comparable to y for calculation of the residual. We can model this relation as

$$y \approx BF\tilde{x} \tag{5.11}$$

Similarly an *upsampling* operator can also be introduced, which is formally the transpose, B^T, such that $BB^T{=}I$. These operators are easy to visualize when B and B^T act upon a vector. For example, the matrix below downsamples a 6-dimensional vector to 3-dimensions,

$$B = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix}. \tag{5.12}$$

In practice, however, there is no need to create these matrices, we simply implement an operation which doubles each pixel (in the 2x super resolution case) along both dimensions and therefore grows the image. The inverse operation shrinks the image by a binning along each dimension, gathering up the flux that was spread between pixels in the upsampling operation. Each upsampling and downsampling operation is normalized such that the total sum of the fluxes stays constant.

The addition of super resolution significantly increases the quality of our result; many small sources which in standard resolution resolve to small jagged/aliased groups of pixels, become uniform in shape and better defined. Higher resolution also allows us to more precisely measure sizes and distances between objects.

5.3.3 Assembling the Pieces

The overall method proceeds as shown in Algorithm 2. For every observation, y_t, we initialize a new PSF and load any associated masks. We then iteratively estimate a new PSF, while holding our image model constant. During this iteration, we apply the aforementioned Robust Statistics weighting, repeatedly updating the our PSF estimate until we've reached convergence. We use two criterions to measure the PSF convergence, an absolute, which quantifies the per-pixel change by calculating a root-mean-square of the differences between the current and previous estimate, and a relative, which ensures we stop once the maximum relative per-pixel change drops below 0.01%. We stop the convergence once either of these criterions are violated.

Once we have an acceptable solution, we compute the update for the image model, while in turn holding the PSF constant. The image model is only updated once per iteration, again we apply our Robust Statistics weighting and our Update Clipping to prevent extreme updates from having adverse effects on our model image. Since the resolved PSFs can be wildly different between observations, the only piece of information carried from one observation to the next is the image model.

Input : Repeated exposures, $\{y_t\}$
Initialize latent image $\tilde{x}$ and the exposure masks;
for *each new exposure* y_t **do**
  Initialize PSF $\tilde{f}_t$
  while *not converged* **do**
    Improve $\tilde{f}_t$ using robust statistics;
  end
  Solve for update u_t with robust weights;
  Apply clipping for stability of sky pixels;
  Update estimated latent image $\tilde{x}$;
end
Output: Latent image and all PSFs

Algorithm 2: Robust blind deconvolution

5.3.4 Application to SDSS

As mentioned previously, we apply our method to repeated observations of SDSS's Stripe 82. We further restrict our study to the region where Stripe 82 overlaps with the Canada-France-Hawaii Telescope Legacy Survey (CFHTLS), featuring a much larger telescope therefore yielding a much deeper image. This allows us to make comparisons not only to the SDSS coadd but also the CFHTLS coadd by Hudelot.[52] CFHTLS has

nearly double the angular resolution of SDSS (SDSS pixel size 1px=0.396", CFHT pixel size 1px=0.187"), making it an ideal candidate for comparison of our super resolution results. CFHTLS enables us to verify our results acting as a sort of *ground truth*. The particular area we chose yields 68 usable observations. All comparisons figures shown on a log scale to highlight robust performance even close to the sky background and around faint objects. For fairness of comparison we scale all our images within each figure to the same total flux, contrast and bias.

In combination, the previously described methods allow us to produce superior results far exceeding the SDSS coadd in quality and even more impressively the results of the CFHTLS coadd. In this section we will discuss our results and the parameters used to achieve them. Good masking is the basis for all of our results, we generate masks as described in section 5.3.1. The edges are then softened using a 15 pixel Gaussian blur preventing edge artifacts. The next ingredient is our robust statistics weighting, section 5.2.1. We experimented with a large variety of different values for the tuning parameter and found $T = 6$ to be a reliably good. The larger this parameter, the smaller the impact of the robust statistics weighting, conversely as this parameter approaches 1, more and more of the image will be affected and potentially causing bright sources to be suppressed.

The sole application of the robust statistics weighting reduces the number of artifacts and noise around and between objects, while also resolving a greatly improved PSF, see *bottom-left* of fig. 5.5. Notice the lack of halos around the objects, as well

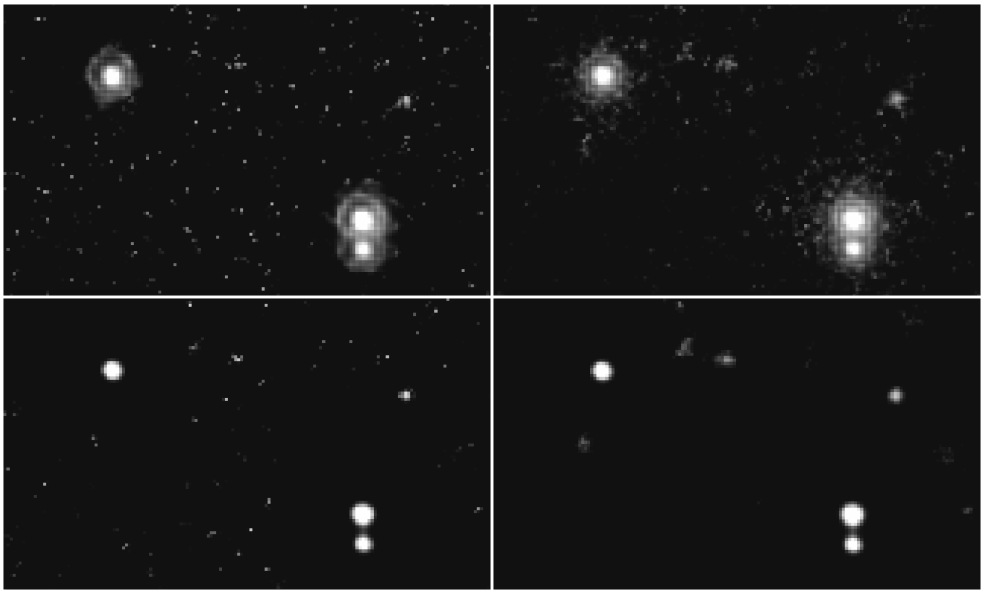

Figure 5.5: unmodified multiframe blind deconvolution (MFBD) (*top-left*), MFBD with Update Clipping (*top-right*), MFBD with robust statistics weighting (*bottom-left*), MFBD with robust statistics weighting and update clipping (*bottom-right*). The clipping removes much of the background noise, but does not have a large effect on the PSF. Robust Statistics weighting provides a much improved PSF and the reduction of noise around objects.

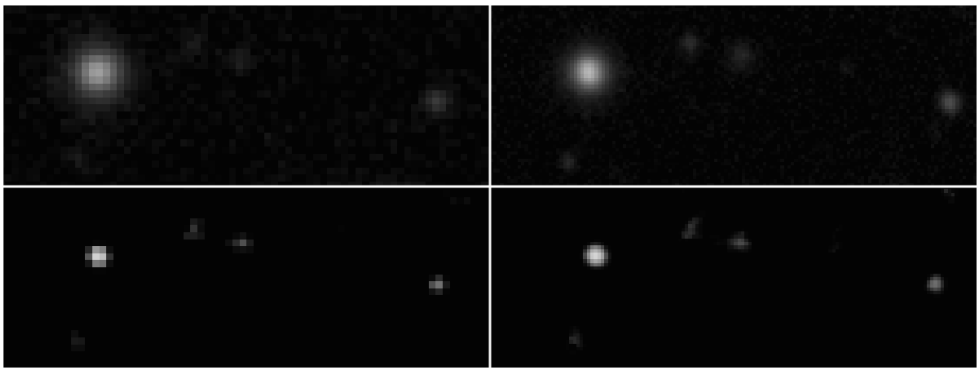

Figure 5.6: Our standard (*bottom-left*) and super resolution (*bottom-right*) results show a clear improvement in Signal-to-noise as well as a much smaller PSF as compared to both the SDSS Coadd (*top-left*) and CFHTLS Coadd (*top-right*). Our super resolution result produces images in comparable resolution and detail to CFHTLS, which is an impressive achievement given that CFHTLS is a much deeper survey with more than double the resolution.

as the reduced number of the background speckles.

To further reduce these background artifacts and prevent erratic updates from disrupting fainter sources, ie. the top center region of the image, we introduce convergence control using the update clipping, section 5.2.2. In our testing we found $d = 2.0$ to be a good clipping threshold. On its own (*top-right* in fig. 5.5), the clipping clears up the vast majority of background speckles.

In combination these methods produce clean and sharp images, which far exceed the quality of the SDSS coadd, which was generated using a similar set of input image, and even exceeding the quality of the CFHTLS coadd. For a fair comparison we show an excerpt of our results with and without super resolution enabled to match the SDSS and CFHTLS coadds, see fig. 5.6.

Point sources are less challenging to deconvolve than complex sources, galaxies are

a good benchmark for validating the quality of our PSFs, as there is more structure that would otherwise get washed out by a poor PSF. In fig. 5.7 we show our robust performance on a spiral galaxy, note the additional detail available on the spiral arms, also note the similarity in the structures of the CFHT coadd.

5.3.5 Software Implementation and Performance

An important consideration with approaches like these is their performance. If a great method takes many hours to process a set of small images, it may yield great results, but it is impractical in a real world applications. For that reason part of our focuses has been on implementing GPU-accelerated methods for all of our compute intensive tasks. Originally we planned on implementing both a full CPU and GPU code path for all features, but as we began to investigating larger (2k by 2k) images, especially with super resolution, it became apparent that a CPU implementation would be too slow to be useful.

Our implementation is written in Python, heavily relying on the use of numpy,[53] a fast numerical and array library, and pyCUDA,[54] a Python interface to NVIDIA's GPU-programming language CUDA.[55] Python, while perhaps not quite as performant as C or C++, offers access to an incredibly rich and mature environment of existing scientific libraries, making development and experimentation a pleasure. However, while investigating the addition of wavelet filtering, we found a severe lack of functional GPU accelerated wavelet transform libraries. At the time of writing there

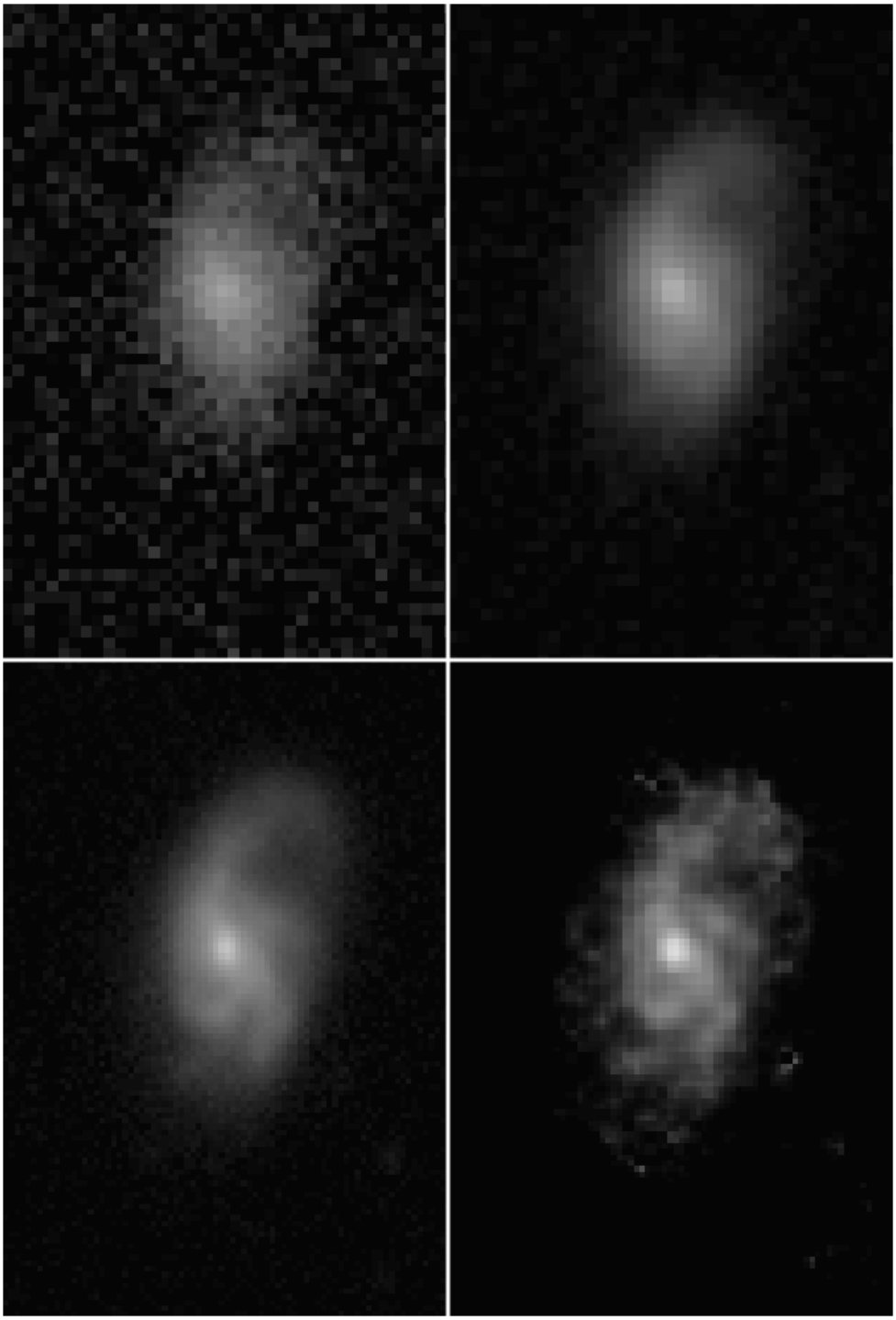

Figure 5.7: Deconvolution of complex sources such as galaxies are a good benchmark of how well a PSF is formed. Here we compare our result (*bottom-right*) against a typical input frame (*top-left*), as well as the SDSS (*top-right*) and CFHTLS (*bottom-left*) coadds.

seems to be only one GPU-accelerated python wavelet transform library, that being PyGASP,[56] which wildly under performs expectations, so much so that the well-known CPU-based python wavelet library, pywt,[57] outperformed PyGASP by 5x on the same input. While we ultimately omitted wavelet filtering from the scope of this research, we believe it may be beneficial and is planned as future work.

Our current implementation can process 140 images (2k by 2k) using standard resolution in under 5 minutes and using 4k by 4k super resolution in approximately 10 minutes, testing performed on an NVIDIA K20 GPU. Of course different parameter settings as well as varying quality of input image does affect the processing time. The times given here are based on our testing completed with SDSS images.

5.4 Future Work and Summary

While our method performs well in many scenarios, we have identified a variety of issues to be addressed as future work. In order for our method to perform at it's best, we rely on a uniform and known background level. Images with poor background subtraction and/or non-uniform backgrounds can impact the final result by wrongly adding or removing flux. At the moment these effects can be largely mitigated by aggressively clipping our updates. Preferably we would solve for a multi-variate background along side the PSF (as an additional set of parameters), allowing us to appropriately remove the background. This would further improve our results, by letting

us dial back the clipping as well as allowing us to include images that previously had to be discarded for large background gradients. Another area of exploration is filtering and smoothing of the residual in order to dampen noise. Initially we figured an approach based on Starck and Murtagh's wavelet filtering method[58] would work well, but after the disappointing results with painfully slow wavelet transform libraries, we decided to revisit this at a later time.

Additionally future work will include validating our results against other surveys such as the Dark Energy Survey (DES)[59] and the Hubble Legacy Archive (HLA)[60,61] verify the quality of our results. Also as part of our future work we plan on evaluating our results in terms of changes to astrometric and photometric errors. First results look promising, but more investigation is needed.

Our method shows a vast gain in quality over Coadds, lucky imaging and the standard Multiframe Blind Deconvolution. The robust statistics weighting successfully prevents outliers to overtake the cost function, therefore providing more realistic PSF estimates and much reduced background noise, the update clipping is shown to prevent artifacts in noise dominated areas, as well as preventing erratic updates from breaking up faint sources. In combination with Super Resolution our methods outperform the current state-of-the-art for combining images, allowing us to produce images that exceed even coadds of a much larger telescope with twice the spacial resolution. We show how to produce robust results from real data in a timely manner. Our method will be instrumental in tackling the colossal datasets produced by

current and upcoming surveys, enabling the production of higher quality and depth of observations than initially available from the instrument itself.

Chapter 6

Processing Pipeline: Scaling Image Deconvolution to Stripe 82

As our datasets continue to grow at unprecedented rates, it is evermore important to not just develop algorithms capable of extracting meaning, but also frameworks which allow us to scale these algorithms to systems large enough to tackle these tasks. While at first glance it may seem trivial to scale across a cluster, the devil is in the details. A large part of the difficulty comes down to obtaining, combining and preprocessing the data into bit-sized chunks which can be fed to the algorithm. The other difficulty is managing the parallel execution of the algorithm. In this chapter we focus on the implementation and execution of the pyMFBD pipeline, a simple yet powerful framework which prepares and processes SDSS's Stripe 82 using the Multiframe Blind Deconvolution tool described in the previous chapter.

6.1 Architecture Overview

When building a pipeline it is important to keep in mind usability, flexibility and portability. This is especially important if such a pipeline should function in various different computing environments. In this section we introduce an architecture of the wrapper and enqueue scripts. These have been designed to expose an easy to understand and use framework for processing large image datasets such as Stripe 82 dataset. While this pipeline have been developed specifically for the Maryland Advanced Research Computing Center (MARCC) and the SDSS Stripe 82 dataset, it is designed to also work with other similarly organized datasets, clusters and image processing tools.

6.1.1 The MARCC Cluster

At the time of writing Maryland Advanced Research Computing Center's (MARCC) main cluster, *bluecrab*[1], consists of 864 nodes, sporting over 21,000 CPU cores and over 700,000 GPU cores (144 NVIDIA Tesla K80). Most of our usage focuses on the GPU nodes, which feature either dual 12 or dual 14 core Intel CPUs, 128GB RAM and 2 NVIDIA Tesla K80s (2x 2496 cores per K80). These nodes are all attached 15 PB (18 PB raw) of ZFS storage. At it's first peak, MARCC's *bluecrab* reached 171st rank on the TOP500 supercomputers list[2].

[1] https://www.marcc.jhu.edu/cyberinfrastructure/hardware/
[2] https://www.top500.org/system/178621

6.1.2 Enqueuing and Job Execution

As alluded to above, there are two distinct components to this pipeline, a wrapper script and an enqueue script. The wrapper script exposes a standardized command line interface. In keeping the wrapper generic, by only requiring the most basic set of command line options, any kind of computation or processing can be executed with this pipeline. While the main task we have been focusing on is deconvolution, we have also created wrapper scripts which can create coadds or execute the required preprocessing as described in the next section. A sample of the wrapper script is shown in Figure 6.2. It only requires 3 arguments, an input and output directory along with a configuration file. In this example the expected configuration file is simply formatted as a *new-line* separated list of *key=value*-pairs, which is expected python's ConfigParser module. A long as the command line options match, the entire implementation of this script is up to the user.

The wrapper script should be a stand along entry point into whichever computation or processing should be achieved. In the simplest case this wrapper script can be executed locally for testing and development but more broadly, when a distributed (scaled up) execution is desired, a pipeline enqueue script is used.

The enqueue script has two main types of executions, one which executes each task locally in series (using `-run_local`) and one which dispatches the executions to remote cluster nodes. The enqueuing tool allows the user to quickly enqueue a large number of job executions, taking all of the hassle out of starting a set of runs.

```
./pipeline_enqueue.py -i ~/data/Stacks \
                      -o ~/data/Processed \
                      -s ./mfbd_wrapper.py \
                      -c ./mfbd.config
```

Figure 6.1: Sample pipeline execution command line arguments

This tool has a small set of required arguments similar to that of the wrapper script, see Figure 6.1, the only difference here is that we specify which wrapper script will be used, as well as having the option to specify `-walk_subdirs`, which will scan of the provided input directory, launching one job execution per contained directory. This provides a quick shortcut for processing entire datasets. Additionally command line options for setup and teardown operations are included, these come in handy when the environment on the cluster node requires some preparation or cleanup with every step. On MARCC we need to use a setup command for loading python at the beginning of the execution, therefore specify: `-setup_cmd='bash -c"module load python"'`.

6.2 Preprocessing and Execution

Before we can run the pyMFBD deconvolution, we need to prepare the image data. The expected format is quite simple, for each portion portion of the sky, a folder containing a stack of pixel-aligned images and their appropriate masks is expected. The masks should cover all known artifacts, such as saturated pixels, CCD

```
#!/usr/bin/env python
import argparse
import ConfigParser
import os
import json

def load_config(config_file):
    c = ConfigParser.ConfigParser()
    c.read(config_file)
    return dict(c.items("DEFAULT"))

def run(input_dir, output_dir, config_file):
    '''Example of wrapper script which will start the execution'''
    if not os.path.exists(output_dir):
        print("Creating output dir: {}".format(output_dir))
        os.mkdir(output_dir)

    with open(output_dir+"/run.log", "w") as log:
        log.write("Starting logging..\n")

        config = load_config(config_file)
        log.write("Provided Config:\n")
        log.write(json.dumps(config)+"\n")

        # Do some work HERE

        log.write("Stopping logging..\n")


parser = argparse.ArgumentParser()
parser.add_argument("-o", dest="output_dir", default="~/results",
                    help="Output directory (Default: ~/results)")
parser.add_argument("-i", dest="input_dir", default="~/data",
                    help="Input directory (Default: ~/data)")
parser.add_argument("-c", dest="config", default="./config.cfg",
                    help="Configuration file (Default: ./config.cfg)")

args = parser.parse_args()
run(args.input_dir,
    args.output_dir,
    args.config)
```

Figure 6.2: Sample wrapper script for pipeline execution

errors, satellite streaks or NaNs.

6.2.1 Stripe 82 Data Access

All of the data required for the processing of Stripe 82 can be access though SDSS's CasJobs interface[3]. We require 3 pieces of information in order to accomplish our processing, the image data, metadata needed for alignment and locations of bright sources needed to be masked out.

Stripe 82 of the Sloan Digital Sky Survey is a 270 square degree area observed along the equatorial plane, with approximately 70 fold coverage. Most of SDSS's observations are taken in drift scan mode, meaning the telescope is pointed to the sky and the rotation of earth effectively scans the across the sky. This generates long continuous width (2048 pixel) images known as *runs*, each of which is chopped into 2048 by 1498 pixel sub sections called *fields*. Fields are created with a 128 pixel overlap between adjacent image fields and stored in the database.

In the following examples we are limiting our dataset to be between within the following Right Ascension (α) range, $330^\circ < \alpha < 360^\circ$. In practice this range can be arbitrarily large, but the majority of the repeat observations can be found in this range. We retrieve the 3 required pieces of data in two steps. The first query, see Figure 6.3, retrieves both the location of all image files as well as all the associated metadata needed in the following steps to align, stitch and cutout the images. Note

[3]http://skyserver.sdss.org/casjobs/

```
SELECT f.run,
       f.camcol,
       f.field,
       dbo.FrameAll.strip,
       dbo.fMuFromEq(f.raMin,f.decMin) as mu1,
       dbo.fMuFromEq(f.raMax,f.decMax) as mu2,
       dbo.fNuFromEq(f.raMin,f.decMin) as nu1,
       dbo.fNuFromEq(f.raMax,f.decMax) as nu2,
       a_r, b_r, c_r,
       d_r, e_r, f_r,
       dbo.fGetUrlFitsCFrame(f.fieldID,'r') as url
FROM dbo.Field as f
  JOIN dbo.FrameAll ON f.fieldID = dbo.FrameAll.fieldID
WHERE f.raMin > 330
    AND f.raMax < 360
    AND f.run != 106
    AND f.run != 206
```

Figure 6.3: Query for retrieving image files and associated metadata

```
SELECT DISTINCT field, skyVersion, run, rerun, camcol,
            objid, ra, dec, psfMag_r, petroMag_r
FROM dbo.PhotoObjAll
WHERE (ra > 330 AND ra < 360 AND petroMag_r < 15)
    AND (flags & dbo.fPhotoFlags('SATURATED')) > 0
```

Figure 6.4: Query for retrieving bright or saturated sources for mask generation.

that this query excludes run 106 and 206, these are Coadd runs stored in the same database. The second query, see Figure 6.4, finds a list of all bright sources or saturated source, which will be used for generation of masks later on.

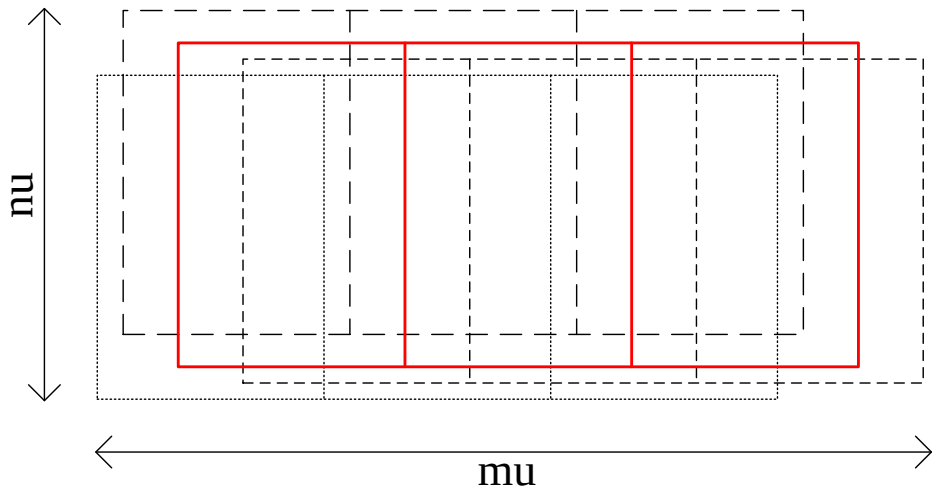

Figure 6.5: Alignment between multiple runs. Each interrupted line style represents fields from a different run. The solid red outline is the set of images we cutout across all available runs.

6.2.2 Alignment, Stitching and Cutouts

There is a fair amount of variability in the alignment of individual drift scan observations (runs) as well as the fields derived from these runs. Figure 6.5 shows an example of three overlapping runs (interrupted outlines), each shifted along mu and nu relative to cutout location (solid red). Fortunately these misalignments are only two dimensional shifts and do not include any rotational differences, this allows for easy x-y translation. In order for us to create a pixel aligned stack of images for processing, we need to align the runs, stitch together nearby fields within a run and finally make pixel-aligned cutout across all runs.

SDSS makes use of multiple different coordinate systems. Of most interest to

us are the Great Circle (mu, nu), the Celestial (ra, dec) and the image-local (pixel) coordinate system. At data retrieval time, we convert all of the Celestial coordinates into survey specific Great Circle coordinates. Additionally we also query for the astrometric calibration constants (a, b, c, d, e, f) needed for the conversion between Great Circle and image-local coordinates.

Alignment of fields from the same run is trivial, simply join adjacent fields while removing the 128 pixel overlap, we refer to this as a stitched run segment. Aligning segments from different runs is a bit more complex and requires the use of the astrometric calibration constants, the great circle coordinates as well as the equations 6.1 and 6.2. Given these we can solve for relative pixel-wise offsets between run segments.

$$\mu = a + bx + cy \qquad \nu = d + ex + fy \tag{6.1}$$

$$x = \frac{f\,(\mu - a) - c\,(\nu + d)}{fb - ce} \qquad y = \frac{e\,(\mu - a) - b\,(\nu + d)}{ec - bf} \tag{6.2}$$

For the pipeline we aim to create pixel-aligned cutout across all available runs in a particular location. First we must choose which location we want to cut out. To do this we select a mu,nu coordinate range, then find all fields which overlap that range, followed by stitching together all fields within each run, creating a run segment. These run segments are then aligned by calculating pixel-wise offsets, which are then used to direct where the cutout is taken from.

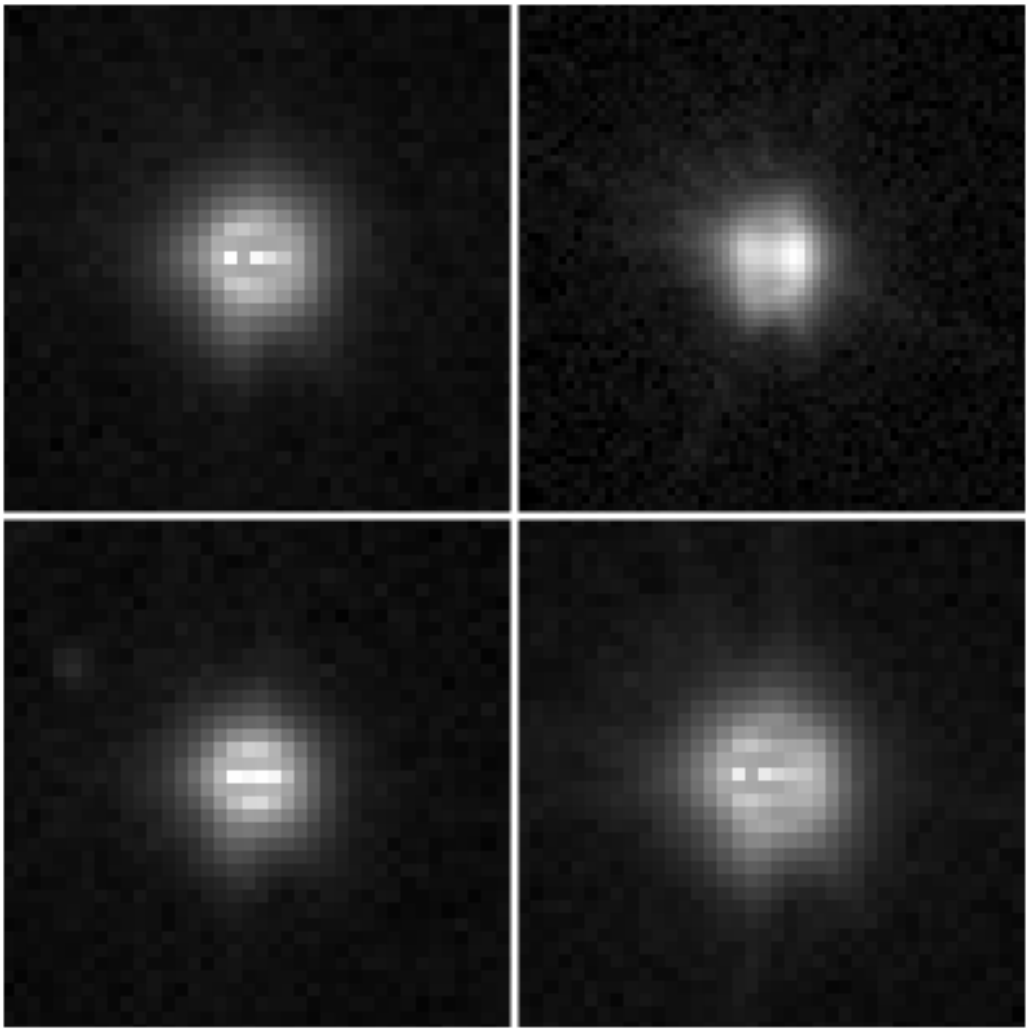

Figure 6.6: PSF anomalies found throughout Stripe 82.

In practice, we cutout the field which correspond to the Annis et al.[2] coadd. This allows for easy comparison and evaluation of our results. In the case that a run only contains a partial match, due to misalignment or missing data, we mask out that area during out mask generation process.

6.2.3 Dealing with Defects

Telescope image can contain a variety of different artifacts. With Stripe 82 in particular, where only 1/4 of the observations were taken under photometric conditions[4], artifacts such as gradients are a common problem. Another issue are objects which contain saturated pixels or other PSF anomalies such as deflection spikes or processing errors resembling death stars (see Figure 6.6). The presence of any of these artifacts can have detrimental impacts on our deconvolution algorithm and therefore must be masked out for the PSF estimation step. To do so, we generate masks of all bright (petroMag < 15) or saturated sources. The location of these sources are retrieved from the database as described in section 6.2.1 and then turned into masks stored along side the cutouts.

Gradients caused by bad seeing are also an artifacts which need to be removed for our deconvolution process. At the moment our image generation pipeline uses SExtractor[62] to estimate the background gradient present in each image. Based on predetermined thresholds we then reject images in which gradients are detected. This would be an interesting area of future exploration. Simple subtraction of the SExtractor-estimated background would be a start, but given the sensitivity of background anomalies, this could introduce unintended artifacts.

[4] http://classic.sdss.org/legacy/stripe82.html

6.2.4 Setting Gears in Motion

Starting a pipeline run is very simple once the preparation work has been completed. As previously described we use the pipeline enqueue script to enqueue processing of all cutouts on the MARCC cluster. For testing of the pipeline we chose to process the northern strip of Stripe 82 in the RA range of $330^\circ < \alpha < 340^\circ$ in the r-band. Preprocessing of this portion of Stripe 82 resulted in a total of 67 stacks. Due to artifacts and gradients present in some of the stacks, we eliminated 4 stacks to ended up with 63 stacks of 64-71 repeat observations each. After all of the preprocessing had been completed, using the following command line options,

```
./pipeline_enqueue.py -i ~/data/Stacks \
                      -o ~/data/Processed \
                      -s ./mfbd_wrapper.py \
                      -c ./mfbd.charlie \
                      --setup_cmd='bash -c "module load python"' \
                      --walk_subdirs
```

we launched the processing pipeline on MARCC. The total elapsed clock time for this particular test run took 49 minutes from launch to completion of the last job. The total time varies depending on the utilization of the cluster, but during our development and testing has usually been less than one hour. Individually, each of the jobs takes between 850 to 1200 seconds to complete. The job execution times varies based on the number and the quality of the input images, as both of these effect convergence.

6.3 Summary and Future Work

As the pipeline stands today, we have created a robust tool allowing for easy development and exploration of large scale image processing tools such as the pyMFBD tool. We have tested this tool on the SDSS Stripe 82 and have shown it's ability to scale to larger datasets. These pipeline tools will be made available along with the rest of the pyMFBD tools on github.

During the development and the application of the pipeline, we discovered multiple possible future improvements. For example, gradients in images pose a difficult problem, currently we are detecting these issues during preprocessing and rejecting the images before they are processed by pyMFBD. Ideally this detection would happend on-the-fly and would result in a correction of the input image instead of a rejection. This would be especially helpful in regions of Stripe 82 which contain very bright sources causing a gradient in every observation. Currently this limits the amount of data we can process.

Another future improvement would be more smarter and more robust masking of artifact-containing objects. It seems the sources we identified as *death stars* are not necessarily marked as bad or saturated in SDSS, additionally we found it difficult to reliably mask all objects containing deflection spikes. Further more, a robust metric for quality would allow for better automatic evaluation and possible rejection of input images, therefore resulting in a better deconvolution result.

Chapter 7

DCR: Subband Image Reconstruction

Dedicated telescopes of modern surveys observe the same region of the sky hundreds of times during their lifetime. Due to changes in the atmosphere, these exposures inherently vary in quality. Time-domain and multicolor studies consider these images together and need advanced method to combine them. Image stacking is traditionally accomplished by either coadding all acceptable quality measurements or a small subset of the best few percent, called lucky imaging. Coadding consists of convolving images to a common resolution before addition,[38] combining images with a restricted range of image qualities,[2] which ignores high-resolution spatial information. Lucky imaging uses the sharpest exposures but looses quality in signal-to-noise ratio, hence depth of the observations. In each of these approaches we sacrifice information (e.g., spatial resolution or depth) for computational expediency or to optimize some aspect about the properties of the images (e.g., image quality).

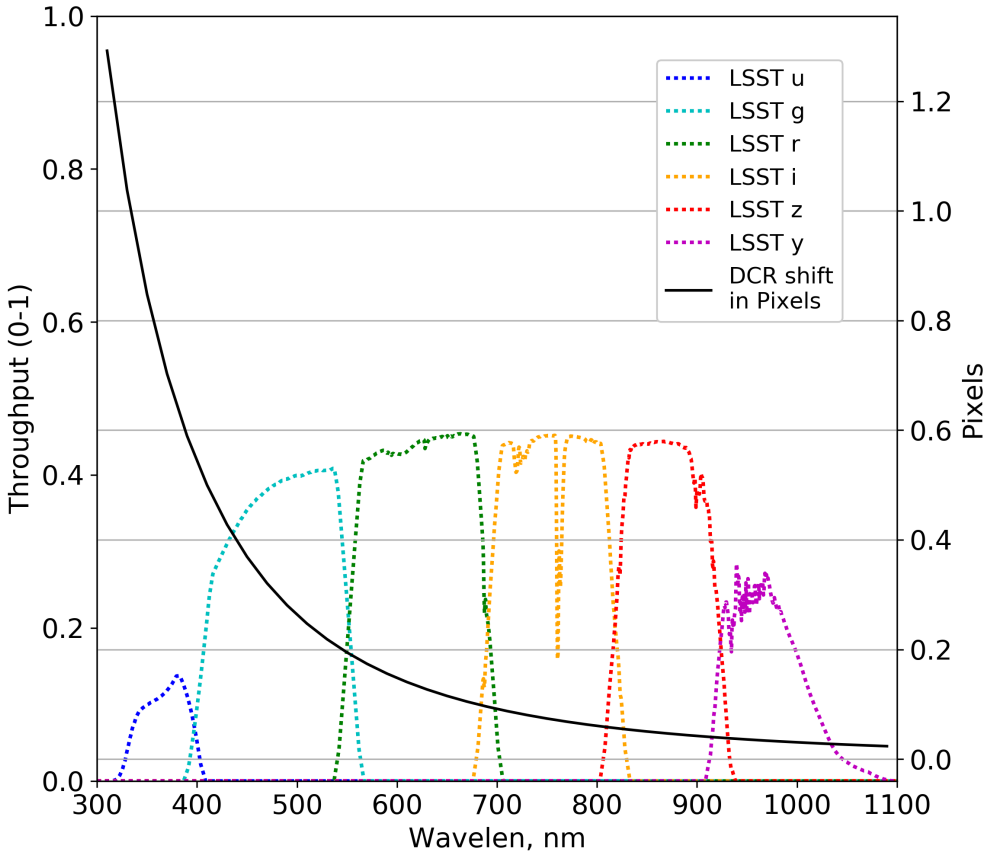

Figure 7.1: The DCR effect is wavelength dependent with lower wavelengths exhibiting a significantly larger astrometric offset. In this plot, we show the effect a mere 10nm difference in wavelengths has (at zenith angle 50) on the center location of a PSF. This offset is measured in units of LSST-pixels (0.2 arcseconds). For reference we also show the LSST bandpass throughputs.

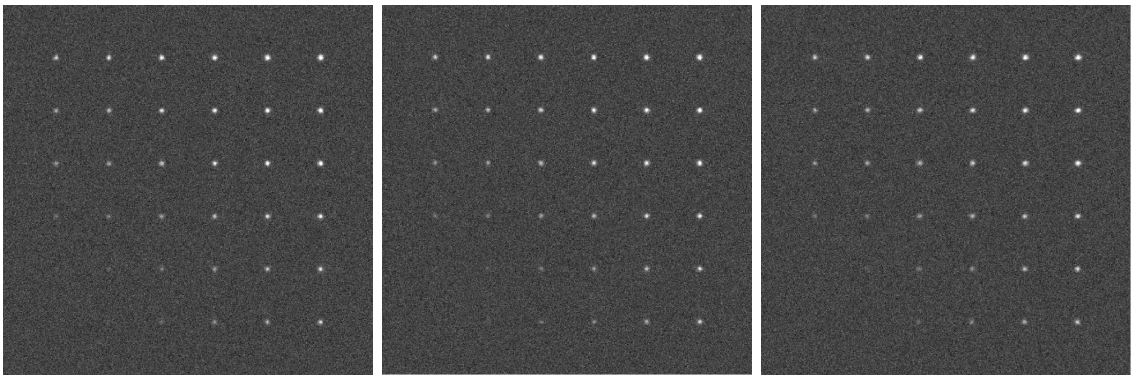

Figure 7.2: Synthetic exposures; *left:* zenith angle 9.85° and azimuth angle 147, observation of 443.7nm and 513.5nm respectively. *center:* zenith angle 25.04° and azimuth angle 334°, observation of 443.7nm and 513.5nm respectively. *right:* zenith angle 50.0° and azimuth angle 77°, observation of 443.7nm and 513.5nm respectively.

In Lee et al.[63] we introduced the concept of learning an underlying model that represents the sky from a sequence of images with differing image qualities and depths. In this approach, an image y_t at epoch t is a simple convolution of the PSF f_t and a latent model image x plus some (uncorrelated) normal noise ϵ_t,

$$y_t = f_t * x + \epsilon_t \quad \text{with} \quad \epsilon_{t,i} \sim \mathcal{N}\big(0, \sigma_{t,i}^2\big) \tag{7.1}$$

where $\sigma_{t,i}$ is the estimated standard deviation of pixel i, i.e., σ_t^2 is the variance map of the y_t image. Since the convolution is a linear operation, in the simplified limit of 1-dimensional vectors, the estimated $\hat{y}_t$ image can be written as a matrix multiplication

$$\hat{y}_t = f_t * x = F_t x \tag{7.2}$$

without loss of generality. Given a set of $\{y_t\}$ exposures and their known $\{f_t\}$ PSFs

we can solve for the model image x in an incremental way by iterating over the observations, considering them one-by-one, and updating the solution using the data from the current, t, epoch. Omitting the t index everywhere for clarity, the multiplicative update formula is

$$x^{(\text{new})} = x \odot \frac{F^T W y}{F^T W \hat{y}} \tag{7.3}$$

where the fraction and the $\odot$ sign are elementwise division and multiplication operations, and the $\hat{y}=Fx$ is the predicted image.[63,64] The W matrix is a combination of the variance map and the binary masks for the exposure including censored areas, e.g., saturated pixels or bad camera columns, and robust weights that arise in modified minimization of dispersion of $r_t = y_t - \hat{y}_t$ residuals in the form of

$$\min_x \sum_{t,i} \rho\left(\frac{r_{t,i}}{\sigma_{t,i}}\right) \quad \text{instead of} \quad \min_x \frac{1}{2}\sum_{t,i}\left(\frac{r_{t,i}}{\sigma_{t,i}}\right)^2 \tag{7.4}$$

In the classical limit of $\rho(t)=t^2/2$, the above two optimization problems are identical, however for other (more) robust ρ-functions, they significantly differ. The typical choice for ρ is quadratic for small residuals, but in case of large discrepancies the function goes to a constant value, say 1, so the outliers' contribution are limited.[51] Such robust minimization essentially corresponds to a Maximum Likelihood Estimation using a likelihood function with a longer tail than the Gaussian to accommodate outliers. Standard solution is an iterative process where we re-weight the quadratic terms with a weight derived from the robust ρ-function, $\mathbb{W}(t)=\rho'(t)/t$. In the classical

case of $\rho(t)=t^2/2$, the weight is constant 1, but outliers have a significant reduction in weight and the Cauchy corresponds to $\mathbb{W}(t) = 1/(1 + t^2/c^2)$;.[51,63]

In this paper, we expand upon these ideas to learn not just the latent image but also to infer the spectral properties of sources from astrometric shifts introduced through DCR. In Section 7.1 we describe our new method capable of extracting color information from sequences of monochromatic images. We apply this technique to simulated images in Section 7.2 where we study the limits of the approach in realistic setting. The results and potential applications of this approach are discussed in Section 7.4.

7.1 Differential Chromatic Refraction

Astronomical sources emit light across the continuum of the electromagnetic spectrum, $S(\lambda)$ with varying intensity. Observed broadband magnitudes or fluxes Y are an integral over the filter $r(\lambda)$ through which the source is observed (including the detector's quantum efficiency).

Considering that the CCD counts photons, the equation is given by

$$Y = \frac{\int d\lambda\ \lambda\ r(\lambda)\, S(\lambda)}{c \int d\lambda\ r(\lambda)/\lambda} \tag{7.5}$$

with c is the speed of light. By substituting for a specific filter

$$R(\lambda) = \frac{\lambda\ r(\lambda)}{c \int d\lambda\ r(\lambda)/\lambda}\,, \tag{7.6}$$

the expression for the flux is significantly simplified,

$$Y = \int d\lambda\ R(\lambda)\, S(\lambda)\,. \tag{7.7}$$

The above equation applies to each pixel in a y_t exposure. The latent image is, however, complicated by differential chromatic refraction (DCR) of the atmosphere. DCR is due to the refraction of light as it passes through the atmosphere similar to light passing through a prism. It results in a positional shift and distortion of sources that depend on its spectral energy distribution.[65] This introduces a point-spread function depending on wavelength, $f_t(\lambda)$, and differing for each epoch t of the observation, based on the zenith and azimuthal pointing of the telescope. Other predictable contributers to this distortion are temperature and humidity of the atmosphere.[66] In addition to these predictable changes, the PSF also varies unpredictably over time due to the turbulence in the atmosphere.

Instead of a latent image x, we introduce the density image $\xi(\lambda)$, and formulate a more general model as

$$y_t = \int d\lambda\ R(\lambda) \Big[f_t(\lambda) * \xi(\lambda) \Big]\ +\ \epsilon_t \tag{7.8}$$

where again the $*$ sign denotes the spatial convolution. In the limit of negligible dependence in the PSF on the wavelength, we get back the previous model

$$y_t = f_t * \left[\int d\lambda \, R(\lambda) \, \xi(\lambda) \right] \; + \; \epsilon_t \; = \; f_t * x + \epsilon_t \tag{7.9}$$

which also provides an interpretation for the latent image x.

If the PSF differs measurably within a passband, simple models including co-adds or simple image deconvolution are inaccurate; as it is the case in the u and g bands of the LSST. In Figure 7.1 we illustrate the strength of this effect as a function of wavelength. The black solid line plotted over the throughput curves represents the amount of relative shift in pixels over a 10nm difference in observation wavelength. For example the differential shift at a zenith angle of 50° across the extremes of the u-band, $\lambda_1 = 320$nm and $\lambda_2 = 400$nm, is over 6 pixels, similarly over the g-band, $\lambda_1 = 320$nm and $\lambda_2 = 400$nm, is approximately 5.5 pixels. Of course in reality, the total apparent shift is dependent on the SED of the source.[67,68] The PSF appears elongated as it consists of a combination all PSF between λ_1 and λ_2.

We, therefore, propose to consider that a latent image is comprised of a number of sub-images each comprising a limited wavelength interval within a given photometric passband. Given the spectral dependence on the PSF, if we can measure an astrometric offset or variation in the PSF as a function of airmass (telescope pointing), we can in principle infer the underlying SED. In these narrower spectral ranges, the

assumptions of a constant PSF and/or flux intensity are more appropriate. We split the filter's wavelength range $[\Lambda_0, \Lambda_K)$ at preset $\{\Lambda_k : k{=}1 \ldots K{-}1\}$ values, and define the response function of these *sub-bands* as

$$R_k(\lambda) = \begin{cases} R(\lambda) & \text{if } \lambda \in [\Lambda_{k-1}, \Lambda_k) \\ \\ 0 & \text{otherwise} \end{cases} \tag{7.10}$$

for all k from 1 to K. Naturally,

$$y_t = \sum_{k=1}^{K} \int d\lambda \; R_k(\lambda) \left[f_t(\lambda) * \xi(\lambda) \right] \; + \; \epsilon_t \tag{7.11}$$

is equivalent to eq.(7.8), and its terms can be approximated similarly to that in eq.(7.9), which yields

$$y_t = \sum_k f_{t,k} * x_k + \epsilon_t \tag{7.12}$$

or

$$y_t = \sum_k F_{t,k}\, x_k + \epsilon_t \tag{7.13}$$

where each x_k is a latent image in the given wavelength range of sub-band k, and $F_{t,k}$ is the linear operator that corresponds to the convolution with the PSF $f_{t,k}$ at the appropriate wavelengths.

Introducing a tall x vector that contains all these x_k (1-dimensional) images

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix} \quad \text{and} \quad F_t = \begin{bmatrix} F_{t,1} & F_{t,2} & \dots \end{bmatrix} \tag{7.14}$$

as the horizontal concatenation of the convolution matrices, we can verify that the equation

$$y_t = F_t x + \epsilon_t \tag{7.15}$$

is equivalent to eq.(7.13). Considering that eq.(7.15) is formally the same as the previous model, cf. eqs.(7.1) and (7.2), the iterative updates of eq.(7.3) directly apply to our new wavelength-dependent model.

Furthermore, considering that the transpose of F (omitting the t index) is

$$F^T = \begin{bmatrix} F_1^T \\ F_2^T \\ \vdots \end{bmatrix} , \tag{7.16}$$

we see that the updates for the individual latent images at different λ_k wavelengths

take the familiar forms of

$$\begin{aligned} x_1^{(\text{new})} &= x_1 \odot \frac{F_1^T W y}{F_1^T W \hat{y}} \,, \\ x_2^{(\text{new})} &= x_2 \odot \frac{F_2^T W y}{F_2^T W \hat{y}} \,, \\ &\vdots \end{aligned} \tag{7.17}$$

and so on. The important difference is that now the common $\hat{y}$ term, the predicted image is the sum of the constituents wavelength dependent image,

$$\hat{y} = \sum_k F_k x_k \,. \tag{7.18}$$

Note that in the limit of a single latent image, we get back the original equations and the corresponding iterative algorithm, hence this is a more general approach With this formalism we can use our robust iterative deconvolution method with only minor modifications.

7.1.1 An Iterative Procedure

This iterative method is comprised of the following steps. We begin by initializing our models, $\left[x_1 \dots x_k\right]$, to be uniform, then start by selecting a random observation, y_t, for which we simulate the appropriate PSF, which we then use in conjunction with equations 7.17, to update our models. We repeat this procedure until we have

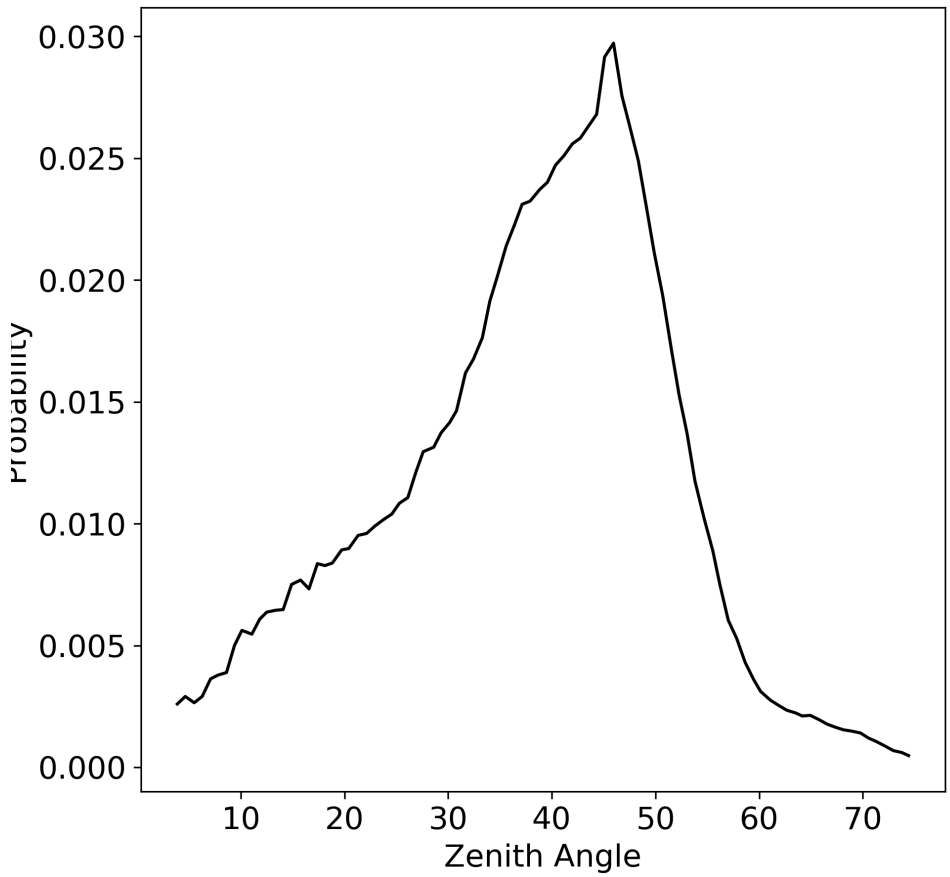

Figure 7.3: The distribution of zenith angles sampled in our simulations, see.[3]

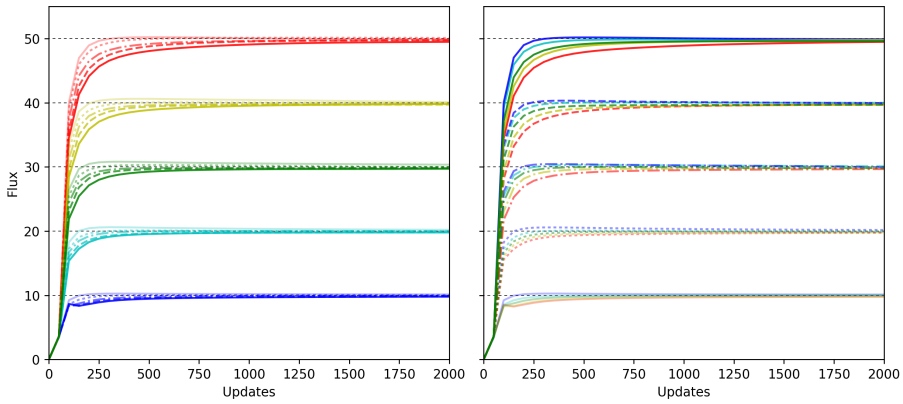

Figure 7.4: Recovered fluxes of objects as function of iteration. *Left Column:* flux 1 as a function of iteration. *Right Column:* flux 2 as function of iteration. The true fluxes for both are a set of 10.0, 20.0, 30.0, 40.0 and 50.0, indicated by the black horizontal guides.

reached convergence. Note that we reuse observations, once we have exhausted our unique supply.

7.2 Synthetic Exposures

To study the properties and limitations of this method we analyze synthetic images with only two components. This essentially corresponds to objects with spectra that only contain two discrete wavelengths. In other words, we attempt to solve for two images at preset wavelengths from a set of exposures that combine these into single passband observations.

Our synthetic exposures are created by observing a latent image through a PSF generated at one of two preset wavelengths. Our latent images are a simple grid of

point sources where the intensity of the individual sub-bands as well as the total flux of the source vary across the image. This simulates sources with varying brightness or color. We will refer to the sub-band images as x_1 and x_2, corresponding to λ_1 and λ_2 respectively.

In order to choose realistic wavelengths, we take the LSST g-band and select the effective wavelengths of the subbands produced by halving the g-band into two equal passbands,

$$\lambda_1 = 443.7\,\text{nm} \quad \text{and} \quad \lambda_2 = 513.5\,\text{nm}. \tag{7.19}$$

To generate a synthetic observation, we choose a zenith and azimuth angle, then generate a PSF for each wavelength at those angles and convolve each latent image with their respective PSFs, finally we add the two resulting images together into our observation. It is important to correctly choose the altitude and airmass of the observations, as these control the strength of the DCR effect at given wavelengths. The zenith angle controls the dispersion and the azimuth controls rotation or direction of the refraction. As the DCR effect is most strongly dependent on the zenith angle, we want to ensure we select a realistic set of zenith angles. To do so we randomly sample the expected distribution of zenith angles, generated based on LSST's OpSim and derived from.[3] See Figure 7.3 for a random sampling of this distribution. Note that the majority of our zenith angles falls between 40° and 50°. This is due to the dome shape of the sky, as the further we are away from the zenith, the more sky can be observed.

For azimuth angles, we randomly sample a uniform distribution of angles ranging from 0° to 360°.

7.2.0.0.1 Synthetic Noise

With the exception of section 7.3.1, we introduce noise to all of our synthetic exposures. More specifically, we add two types of pixel-wise noise, one which is simply drawn from a zero-mean normal distribution and one which is pixel-value dependent, *shot noise*, $\epsilon_i = \varepsilon + \alpha x_i$, where $\varepsilon \sim \mathcal{N}(0,\, \sigma^2)$ and $\alpha = 0.01$. We choose our true fluxes and noise σ such that the sources in our individual exposures range in signal-to-noise from 1 to 9. This allows us to test our method under realistic conditions.

7.2.0.0.2 Generating Exposures

Using the StarFast Simulator,[69] we generate 200 exposures with a random selection zenith and azimuth angles, chosen as described above. We run tests on subsets of these 200 observations(5, 15, and 50) to simulate how well we can recover the DCR corrected images in the early years of the survey. Additionally, when noise is introduced, we generate 50 random realizations of the noise to quantify its effect. Below we analyze these and present our results.

Figure 7.2 illustrates random realization of these synthetic exposures. The sources in x_1 vary from bottom to top, providing approximately 0, 1, 3, 5, 7 and 9 SNR, x_2 contains the same grid but varying from left to right.

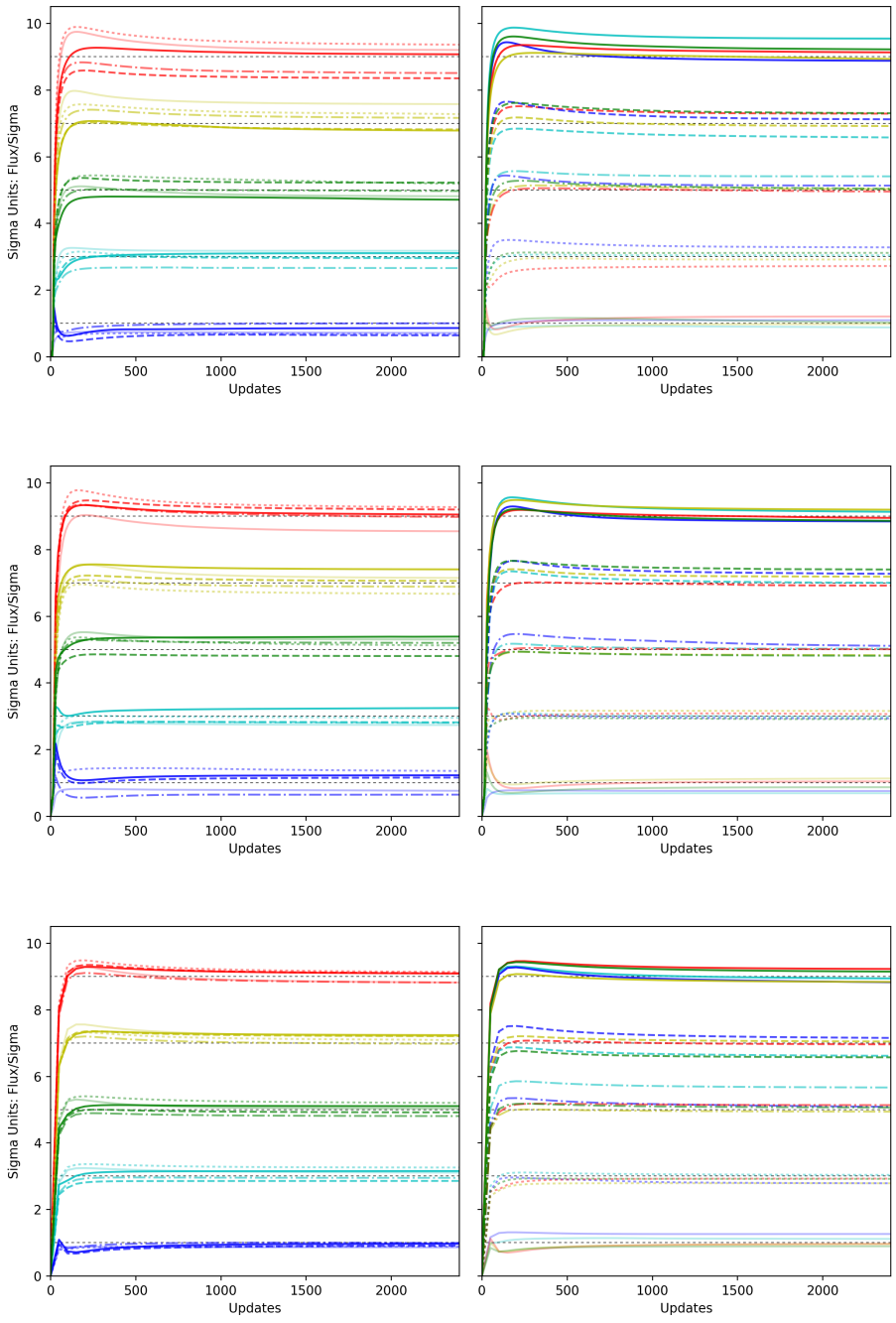

Figure 7.5: Recovered fluxes of objects as function of iteration. *Left Column:* flux 1 measured in units of Sigma as a function of iteration. *Right Column:* flux 2 measured in units of Sigma as function of iteration. The true sigma units for both are a set of 1.0, 3.0, 5.0, 7.0 and 9.0, indicated by the black horizontal guides. *Fist Row:* reconstruction based on using 5 random observations; *Second Row:* reconstruction based on 15 random observations; *Third Row:* reconstruction based on 50 random observations. Each random observations has a different combination of zenith angle and azimuth.

7.3 Quantifying Quality

In the following two sections, we examine the performance of this method in the noiseless case and then evaluate it's performance as we add noise and reduce the number of input observations.

7.3.1 The Noiseless Limit

We first evaluate the performance of this method without the presence of noise. To analyze the relative quality of our deconvolution method, we measure the fluxes of all sources as a function of iteration of our deconvolution algorithm as described in Section 7.1.1.

Figure 7.4 shows the recovered flux values as they converge as a function updates to our image model. For this experiment we used 50 observations, with flux values ranging from 10 to 50 flux units (fu) arranged in a grid as described above. The left panel contains the fluxes extracted from the x_1 image, corresponding to λ_1, and the right panel shows the fluxes extracted from the x_2 image, corresponding to λ_2. Each of the 25 sources present in this plot, is assigned a unique color-linestyle combination which is common between both plots. For example the dotted red line corresponds to a source, which contains 50 fu in x_1 and 20 fu in x_2. All fluxes are separated and recovered within $\pm 4.08\%$ RMS at iteration 300, within $\pm 1.3\%$ RMS at iteration 1000 and within $\pm 0.89\%$ RMS at iteration 2000.

7.3.2 Noisy Exposures

Next we introduce noise and show we retain the ability to separate out the fluxes. Figure 7.5 shows sample results with added noise as well as a varying the number of input observations. The plots from top to bottom correspond to 5, 15 and 50 input observations. For these observations we add a pixel-wise noise with $\sigma = 0.1$, we scale our input fluxes such that our sources end up with an SNR between 1 and 18.

7.3.2.0.1 Flux Accuracy and Consistency

In the presence of noise we see errors in the final recovered fluxes, as expected the more information available, i.e. more observations, the lower the errors. For example, when looking at the random realization presented in Figure 7.5, at update/iteration 300 we observe a $\pm 11.1\%$ RMS with 5 observations, $\pm 12.2\%$ RMS with 15 observations and $\pm 9.4\%$ RMS with 50 observations. As we reach 1000 updates, we observe a $\pm 11.1\%$ RMS with 5 observations, $\pm 11.4\%$ RMS with 15 observations and $\pm 6.7\%$ RMS with 50 observations. This sample is only limited to a single random realization, summarizing all 50 the realizations is the covariance scatter plot shown in Figure 7.6. This plot not only shows the spread variance among the realizations, but also a strong anti-correlation of the flux pairs. Meaning that if a source's flux is measured to be low in x_1 it is likely to be similarly high in x_2 and vise-versa. This is due to the fact that the total flux of that source is well constrained, where as the separation of the fluxes is much more difficult to extract, therefore requiring many iterations of

updates. While the 45° axis shows the anti-correlation, the 135° axis shows the error in total flux between the two passbands.

To summarize the variations between realizations, we look to Figure 7.7, which shows the average bias in the top row and the variance on the bottom row, both as a function of source SNR.

7.3.2.0.2 Positional Accuracy and Consistency

Beyond evaluating the photometery, we also evaluate the astrometric accuracy. In Figure 7.8 and 7.9 we compare positional accuracy across all random realizations between a coadd of the input images and the result of our deconvolution process. The coadd we compare against here is simply the mean of all of our input observations, without any correction for DCR.

In the top row of each figure we show the pixel-wise positional bias extracted from the sources, meaning this is the average offset from its true position. This is plotted as a function of the SNR of the source. The positional bias of the coadd is largely effected by the brightness of the source, fainter sources display a larger bias, for example the 1 SNR displays a bias nearly 10x higher than our results below. On the contrary, our result shows little bias across the board, with an average positional bias below 0.01 pixels. Even more so in the 50 and 200 input image case, where the average bias is below 0.005 pixels. This is largely do to the fact that the deconvolution process deblurs the source, allowing for a more accurate position finding (center of

mass).

Besides positional bias we also evaluate the root-mean-square error (variability) of the bias. Interestingly the error of the bias is very similar for the 200 input image case, but quickly diverges for smaller numbers of input images. In the 15 input image case, the average error across all sources for the coadd is 0.08 pixels, where as for the deconvolution result it's only 0.04 pixels.

7.4 Discussion and Summary

We believe this techniques will be an important tool for correctly combining observations effected by DCR. This is especially relevant to exposures captured in the G and U band at zenith angles most strongly effected by DCR.

While this paper only discusses the applicability of this method using simulated LSST observations, it is certainly conceivable to apply this method to existing surveys, as long as the telescope's pointing direction is available and the effect of DCR is well enough understood to generate subband PSFs.

Another interesting feature of this technique is it's ability to extract spectral information from images. While the main focus in this paper has been on resolving two subbands, this method easily be extended to more subbands. In 7.10, we use our method to resolve 3 subbands given a realistic set of images produced by using LSST's Starfast Simulator.[69] The input images are generated using the full G-band spectrum,

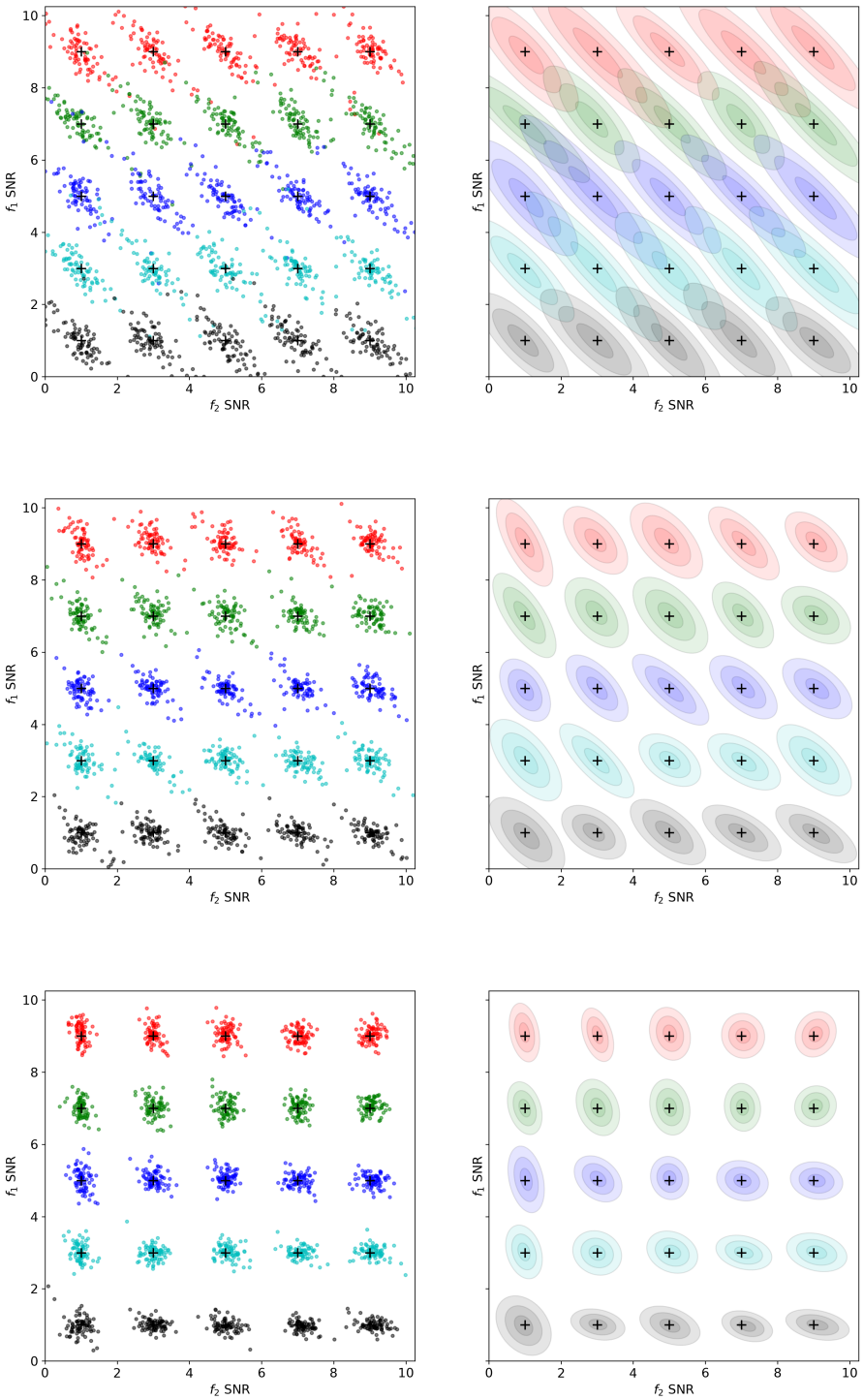

Figure 7.6: *Left:* Scatter plot of resolved fluxes across multiple realizations; *Right:* Covariance distribution for each flux1-flux2 pairing, ellipses drawn at 1σ, 2σ and 3σ

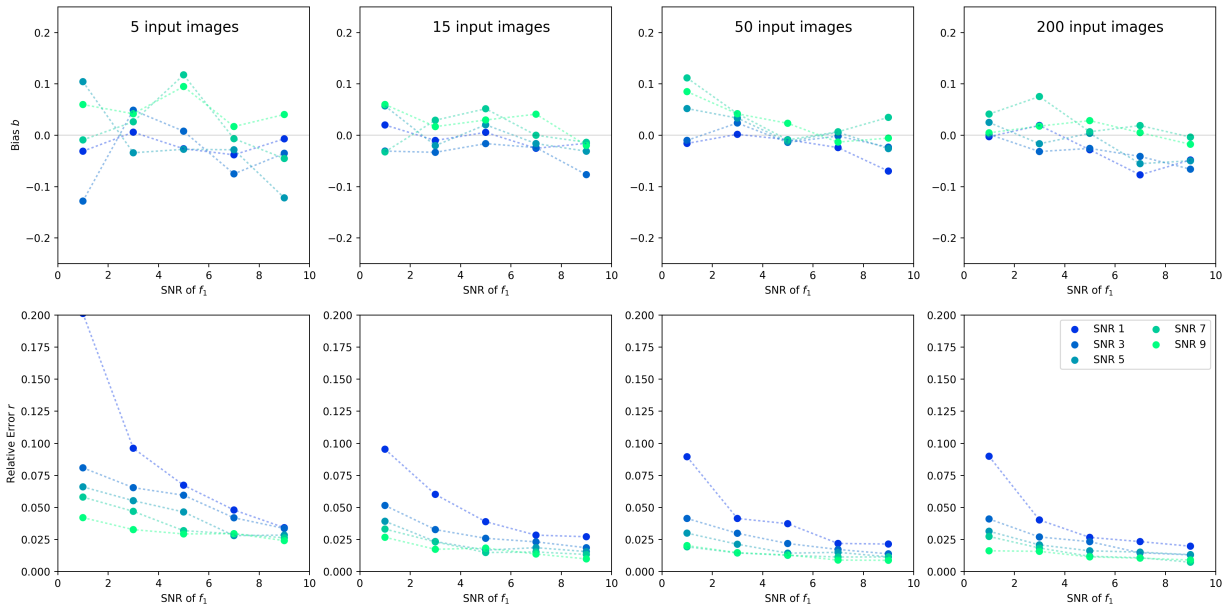

Figure 7.7: Relative Bias and Relative Error of Recovered fluxes.

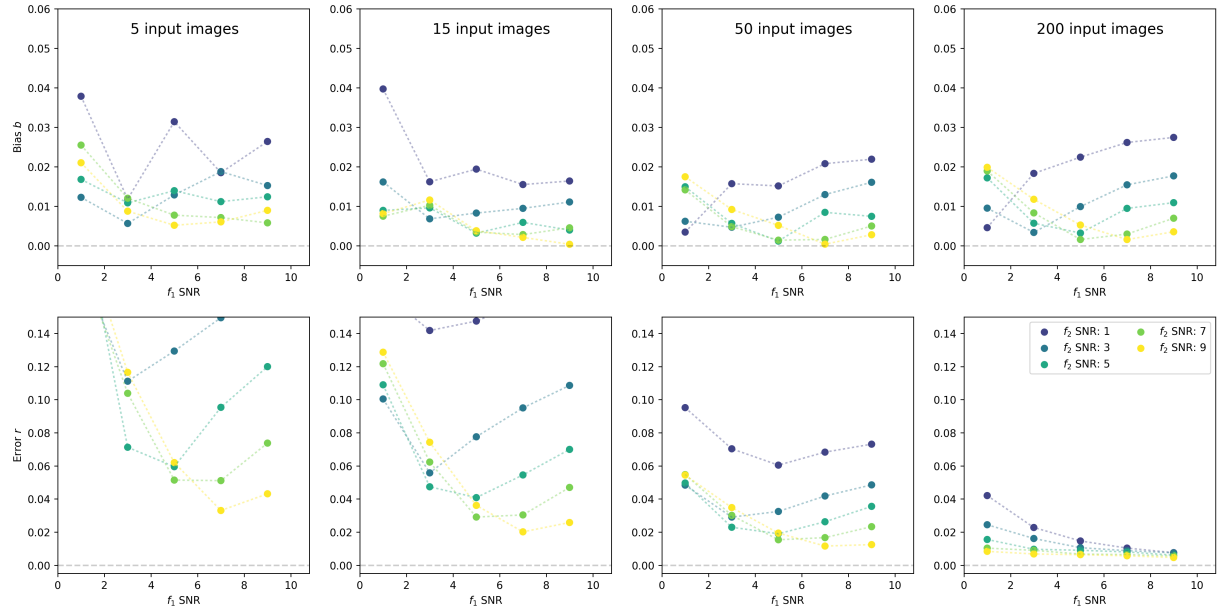

Figure 7.8: Bias of position, coadded input frames

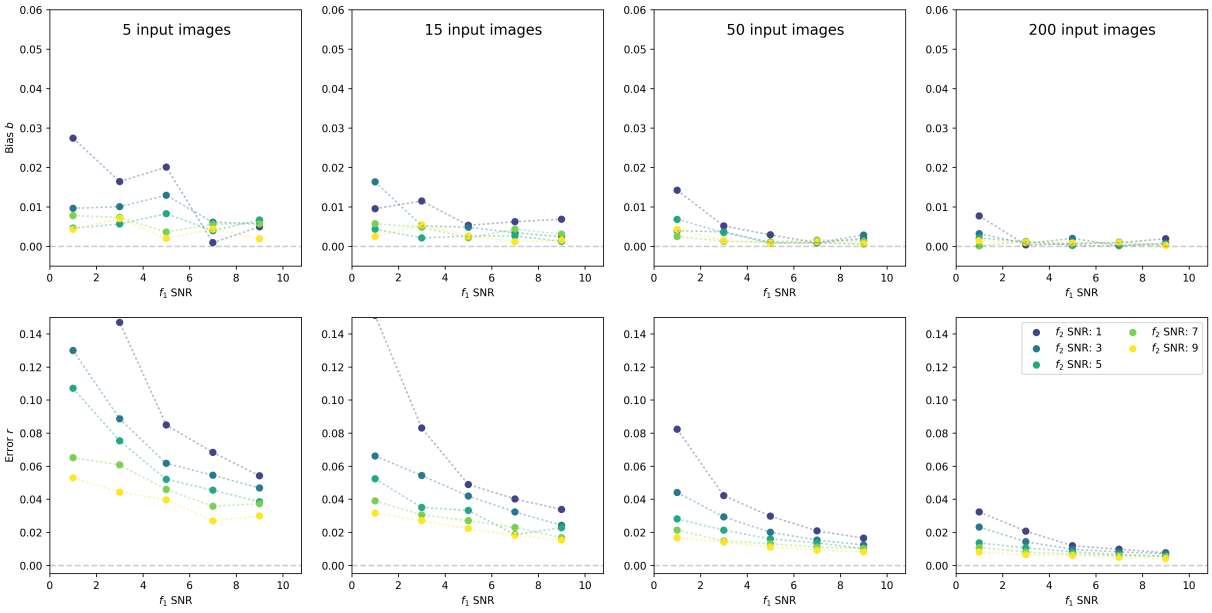

Figure 7.9: Bias of position, extracted from x_1 and x_2

these observations are shown in the *center-column*. In the *left-column* we generate the hypothetical 3-band observation if LSST's subband filter were subdivided into 3 equally wide filters, note that these are only for illustration and are not used in our deconvolution. On the *right*, is our result of processing 200 monochrome g-band observations, similar to those in the *center-column*. Not only do we correctly resolve a all 3 subband images (shown as inverted RGB), but we also significantly deblur the image and correct for the positional error caused by DCR. This is very promising result, is a motivating factor driving out future work, in which we will embark on a full analysis and tuning of full-spectrum DCR deconvolution.

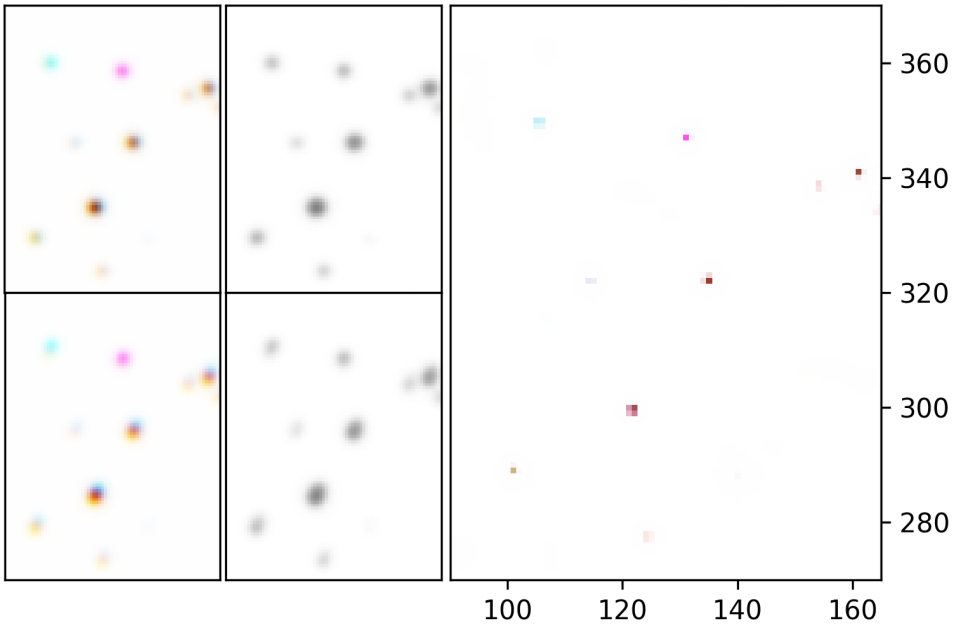

Figure 7.10: *left-column:* two typical sample observation containing a combination of separately observing 3 subbands. Note this is purely in simulation and would not be possible without modifying the telescope. *center-column:* the corresponding monochrome g-band observations, these are simulated full g-band observations. *right:* result of our deconvolution process, resolving the corresponding RGB colors of the *left-column*, using only images similar to the *center-column* as an input. We show a deconvolution result not only resolving a higher color resolution, than the observed images, we also significantly deblur the image and correct for the positional error caused by DCR

Chapter 8

Conclusion

Scalable processing and extraction of meaning from large datasets are crucial to keep up with the exponentially growing influx of data. In this thesis we presented a variety of powerful and novel approaches to enable modern and next-generation science and engineering. We establish the versatility and power of the modern GPU as a tool for taming the ever growing torrents of data. We show that while GPUs have been pioneered and democratized mostly for entertainment, they have made a strong foray into the world of scientific and enterprise computing.

We addressed computational power efficiency by show casing next generation compute hardware using low-power ARM processors as the host system for GPUs. We present results indicating that platforms like these are capable of producing vastly more computations per watt than traditional systems, allowing power strapped datacenter to reduce both machine and cooling power consumption. Given recent advances

of miniaturizing fully CUDA-capable GPUs onto the same die as the ARM SoC, it is likely these systems will become more relevant in the near future, especially as the desire to incorporate deep learning into mobile platforms becomes a viable option.

We proposed and studied new high-performance methods for searching and matching extremely large dataset. The first being a GPU accelerated genomic sequence aligner tuned for aligning long genomic sequences and the second being a multi-GPU astronomy catalog cross matching tool capable of evaluating candidate pairs at a rate of over 1 trillion per millisecond when scaled across 4 GPUs. Methods and tools such as these are enabling technologies for future research, allowing use to explore, search and most critically combine information at very large scales, as not just the size, but also the number of datasets is quickly growing.

Computational optics have existed for many decades, but it has not been until recently that enough computational power was easily accessible in order to truly study such methods. We studied the limits of existing methods and devised a novel powerful approach, combining information across hundreds of extremely large telescope observations, successfully resolving the PSF, deblurring and denoising while also doubling the resolution. This is a truly ground breaking method enabling us to extract vastly more information than previously possible. We built upon this approach, extending it to not only deconvolve hundreds of observations, but also be able to extract color information from monochrome images given only the angle of the observation. We believe this approach will revolutionize modern astronomy as it allows us to extract

color information at a higher resolution than originally available from the physical telescope, and additionally allows us to correct for differential chromatic refraction resulting in more precise measurements.

As a whole we established multiple instances in which we have devised novel methods to enable the next generation of big data science, accelerating search, matching and extraction of meaning from massive datasets. Much of this work has been enabled by the capabilities of the modern GPU in conjunction with advances in computational optics, matching and searching algorithms. This has allowed us to extract more meaning out of vast quantities of data than previously possible, all while accomplishing this computation on a reasonable power budget.

Bibliography

[1] M. Lee and T. Budavári, "Faster catalog matching on graphics processing units," *Astronomy and Computing*, pp. –, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S221313371730032X

[2] J. Annis, M. Soares-Santos, M. A. Strauss, A. C. Becker, S. Dodelson, X. Fan, J. E. Gunn, J. Hao, Z. Ivezic, S. Jester *et al.*, "The sdss coadd: 275 deg^ 2 of deep sdss imaging on stripe 82," *arXiv preprint arXiv:1111.6619*, 2011.

[3] J. Sebag and K. Vogiatzis, "Estimating dome seeing for lsst," in *SPIE Astronomical Telescopes+ Instrumentation*. International Society for Optics and Photonics, 2014, pp. 91 500R–91 500R.

[4] F. NVidia, "Nvidia's next generation cuda compute architecture," *NVidia, Santa Clara, Calif, USA*, 2009.

[5] T. Dong, V. Dobrev, T. Kolev, R. Rieben, S. Tomov, and J. Dongarra, "A step towards energy efficient computing: Redesigning a hydrodynamic application on

cpu-gpu," in *Parallel and Distributed Processing Symposium, 2014 IEEE 28th International*. IEEE, 2014, pp. 972–981.

[6] A. Vasan, A. Sivasubramaniam, V. Shimpi, T. Sivabalan, and R. Subbiah, "Worth their watts?-an empirical study of datacenter servers," in *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*. IEEE, 2010, pp. 1–10.

[7] S. Pelley, D. Meisner, T. F. Wenisch, and J. W. VanGilder, "Understanding and abstracting total data center power," in *Workshop on Energy-Efficient Design*, 2009.

[8] A. S. Szalay, G. C. Bell, H. H. Huang, A. Terzis, and A. White, "Low-power amdahl-balanced blades for data intensive computing," *ACM SIGOPS Operating Systems Review*, vol. 44, no. 1, pp. 71–75, 2010.

[9] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology*, vol. 48, no. 3, pp. 443–453, 1970.

[10] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[11] M. Burrows and D. J. Wheeler, "A block-sorting lossless data compression algorithm," 1994.

[12] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows–wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[13] J. Felsenstein, S. Sawyer, and R. Kochin, "An efficient method for matching nucleic acid sequences," *Nucleic Acids Research*, vol. 10, no. 1, pp. 133–139, 1982.

[14] E. Cheever, D. Searls, W. Karunaratne, and G. Overton, "Using signal processing techniques for dna sequence comparison," in *Bioengineering Conference, 1989., Proceedings of the 1989 Fifteenth Annual Northeast.* IEEE, 1989, pp. 173–174.

[15] A. L. Rockwood, D. K. Crockett, J. R. Oliphant, and K. S. Elenitoba-Johnson, "Sequence alignment by cross-correlation," *Journal of biomolecular techniques: JBT*, vol. 16, no. 4, p. 453, 2005.

[16] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[17] T. Budavári and A. S. Szalay, "Probabilistic cross-identification of astronomical sources," *The Astrophysical Journal*, vol. 679, no. 1, p. 301, 2008.

[18] S. Heinis, T. Budavári, and A. S. Szalay, "Cross-identification performance from simulated detections: Galex and sdss," *The Astrophysical Journal*, vol. 705, no. 1, p. 739, 2009.

[19] G. Kerekes, T. Budavári, I. Csabai, A. J. Connolly, and A. S. Szalay, "Cross Identification of Stars with Unknown Proper Motions," *Astrophysical Journal*, vol. 719, pp. 59–66, August 2010.

[20] T. Budavári and T. J. Loredo, "Probabilistic record linkage in astronomy: Directional cross-identification and beyond," *Annual Review of Statistics and Its Application*, vol. 2, no. 1, pp. 113–139, 2015. [Online]. Available: http://dx.doi.org/10.1146/annurev-statistics-010814-020231

[21] L. Dobos, T. Budavári, N. Li, A. S. Szalay, and I. Csabai, "Skyquery: an implementation of a parallel probabilistic join engine for cross-identification of multiple astronomical databases," in *International Conference on Scientific and Statistical Database Management*. Springer, 2012, pp. 159–167.

[22] T. Budavári, L. Dobos, and A. S. Szalay, "SkyQuery: Federating astronomy archives," *Computing in Science & Engineering*, vol. 15, no. 3, pp. 12–20, May 2013. [Online]. Available: http://scitation.aip.org/content/aip/journal/cise/15/3/10.1109/MCSE.2013.41

[23] T. Boch, F. X. Pineau, and S. Derriere, "CDS xMatch Service Documentation," May 2016.

[24] P. Z. Kunszt, A. S. Szalay, and A. R. Thakar, "The Hierarchical Triangular Mesh," in *Mining the Sky*, A. J. Banday, S. Zaroubi, and M. Bartelmann, Eds., 2001, p. 631.

[25] K. M. Górski, E. Hivon, A. J. Banday, B. D. Wandelt, F. K. Hansen, M. Reinecke, and M. Bartelmann, "HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere," *Astrophysical Journal*, vol. 622, pp. 759–771, April 2005.

[26] R. G. Crittenden, "Igloo Pixelizations of the Sky," *Astrophysical Letters and Communications*, vol. 37, p. 377, 2000.

[27] R. Scranton, M. Tegmark, and Y. Xu. [Online]. Available: http://lahmu.phyast.pitt.edu/âĹijscranton/SDSSPix

[28] J. Gray, M. Nieto-Santisteban, and A. Szalay, "The zones algorithm for finding points-near-a-point or cross-matching spatial datasets," *arXiv preprint cs/0701171*, 2007.

[29] N. Bell and J. Hoberock, "Thrust: A productivity-oriented library for cuda," *GPU computing gems Jade edition*, vol. 2, pp. 359–371, 2011.

[30] C. Nvidia, "C programming guide version 4.0," *Nvidia Corporation*, 2011.

[31] M. Harris, S. Sengupta, and J. Owens, "Parallel prefix sum (scan) with cuda," *GPU Gems*, vol. 3, no. 39, pp. 851–876, 2007.

[32] T. Budavari and M. A. Lee, "Xmatch: GPU Enhanced Astronomic Catalog Cross-Matching," Astrophysics Source Code Library, Mar. 2013.

[33] W. H. Richardson, "Bayesian-based iterative method of image restoration," *JOSA*, vol. 62, no. 1, pp. 55–59, 1972.

[34] L. B. Lucy, "An iterative technique for the rectification of observed distributions," *The Astronomical Journal*, vol. 79, p. 745, 1974.

[35] C. J. Burrows, J. A. Holtzman, S. Faber, P. Y. Bely, H. Hasan, C. Lynds, and D. Schroeder, "The imaging performance of the hubble space telescope," *The Astrophysical Journal*, vol. 369, pp. L21–L25, 1991.

[36] C. C. Cunningham and D. Anthony, "Image deconvolution of extended objects: A comparison of the inverse fourier and the lucy techniques," *Icarus*, vol. 102, no. 2, pp. 307–315, 1993.

[37] J. Krist and H. Hasan, "Deconvolution of hst wfpc images using simulated psfs," in *Astronomical Data Analysis Software and Systems II*, vol. 52, 1993, p. 530.

[38] L. Lucy and R. Hook, "Co-adding images with different psf's," in *Astronomical Data Analysis Software and Systems I*, vol. 25, 1992, p. 277.

[39] J. Nunez and J. Llacer, "A general bayesian image reconstruction algorithm with entropy prior. preliminary application to hst data," *Publications of the Astronomical Society of the Pacific*, pp. 1192–1208, 1993.

[40] D. Homrighausen, C. Genovese, A. Connolly, A. Becker, and R. Owen, "Image

co-addition with temporally varying kernels," *Publications of the Astronomical Society of the Pacific*, vol. 123, no. 907, pp. 1117–1126, 2011.

[41] D. Fish, A. Brinicombe, E. Pike, and J. Walker, "Blind deconvolution by means of the richardson–lucy algorithm," *JOSA A*, vol. 12, no. 1, pp. 58–65, 1995.

[42] S. Harmeling, M. Hirsch, S. Sra, and B. Scholkopf, "Online blind deconvolution for astronomical imaging," in *Computational Photography (ICCP), 2009 IEEE International Conference on.* IEEE, 2009, pp. 1–7.

[43] R. N. Tubbs, "Lucky exposures: Diffraction limited astronomical imaging through the atmosphere," *arXiv preprint astro-ph/0311481*, 2003.

[44] D. G. York, J. Adelman, J. E. Anderson Jr, S. F. Anderson, J. Annis, N. A. Bahcall, J. Bakken, R. Barkhouser, S. Bastian, E. Berman *et al.*, "The sloan digital sky survey: Technical summary," *The Astronomical Journal*, vol. 120, no. 3, p. 1579, 2000.

[45] K. N. Abazajian, J. K. Adelman-McCarthy, M. A. Agüeros, S. S. Allam, C. A. Prieto, D. An, K. S. Anderson, S. F. Anderson, J. Annis, N. A. Bahcall *et al.*, "The seventh data release of the sloan digital sky survey," *The Astrophysical Journal Supplement Series*, vol. 182, no. 2, p. 543, 2009.

[46] J. Starck, E. Pantin, and F. Murtagh, "Deconvolution in astronomy: A review,"

Publications of the Astronomical Society of the Pacific, vol. 114, no. 800, pp. 1051–1069, 2002.

[47] R. L. White, "Image restoration using the damped richardson-lucy method," in *1994 Symposium on Astronomical Telescopes & Instrumentation for the 21st Century*. International Society for Optics and Photonics, 1994, pp. 1342–1348.

[48] S. Harmeling, S. Sra, M. Hirsch, and B. Scholkopf, "Multiframe blind deconvolution, super-resolution, and saturation correction via incremental em," in *Image Processing (ICIP), 2010 17th IEEE International Conference on*. IEEE, 2010, pp. 3313–3316.

[49] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[50] T. R. Lauer, "Deconvolution with a spatially-variant psf," *arXiv preprint astro-ph/0208247*, 2002.

[51] R. Maronna, D. Martin, and V. Yohai, *Robust statistics*. John Wiley & Sons, Chichester. ISBN, 2006.

[52] P. Hudelot, Y. Goranova, Y. Mellier, H. J. McCracken, F. Magnard, M. Monnerville, G. Smah, J.-C. Cuillandre *et al.*, "T0007: The final cfhtls release," 2012.

[53] D. Ascher, P. F. Dubois, K. Hinsen, J. Hugunin, T. Oliphant *et al.*, "Numerical python," 2001.

[54] A. Klöckner, N. Pinto, Y. Lee, B. Catanzaro, P. Ivanov, and A. Fasih, "PyCUDA and PyOpenCL: A Scripting-Based Approach to GPU Run-Time Code Generation," *Parallel Computing*, vol. 38, no. 3, pp. 157–174, 2012.

[55] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.

[56] N. Bowman, E. Carrier, and G. Wolffe, "Pygasp: Python-based gpu-accelerated signal processing," in *Electro/Information Technology (EIT), 2013 IEEE International Conference on.* IEEE, 2013, pp. 1–6.

[57] F. Wasilewski, "Pywavelets: Discrete wavelet transform in python," 2010, http://www.pybytes.com/pywavelets/.

[58] J.-L. Starck and F. Murtagh, "Image restoration with noise suppression using the wavelet transform," *Astronomy and Astrophysics*, vol. 288, pp. 342–348, 1994.

[59] D. E. S. Collaboration *et al.*, "The dark energy survey," *arXiv preprint astro-ph/0510346*, 2005.

[60] H. Jenkner, R. Doxsey, R. Hanisch, S. Lubow, W. Miller III, and R. White, "Concept for the hubble legacy archive," in *Astronomical Data Analysis Software and Systems XV*, vol. 351, 2006, p. 406.

[61] B. Whitmore, K. Lindsay, and M. Stankiewicz, "Source lists for the hubble legacy archive (hla)," in *Astronomical Data Analysis Software and Systems XVII*, vol. 394, 2008, p. 481.

[62] E. Bertin and S. Arnouts, "Sextractor: Software for source extraction," *Astronomy and Astrophysics Supplement Series*, vol. 117, no. 2, pp. 393–404, 1996.

[63] M. Lee, T. Budavári, R. White, and C. Gulian, "Robust statistics for image deconvolution," *Astronomy and Computing*, 2017.

[64] M. A. Lee and T. Budavári, "Streaming multiframe deconvolutions on gpus," in *Astronomical Data Analysis Software an Systems XXIV (ADASS XXIV)*, vol. 495, 2015, p. 261.

[65] J. E. Meyers and P. R. Burchat, "Impact of atmospheric chromatic effects on weak lensing measurements," *The Astrophysical Journal*, vol. 807, no. 2, p. 182, 2015.

[66] A. V. Filippenko, "The importance of atmospheric differential refraction in spectrophotometry." *Publications of the Astronomical Society of the Pacific*, vol. 94, no. 560, p. 715, 1982.

[67] M. C. Kaczmarczik, G. T. Richards, S. S. Mehta, and D. J. Schlegel, "Astrometric redshifts for quasars," *The Astronomical Journal*, vol. 138, no. 1, p. 19, 2009.

[68] C. M. Peters, G. T. Richards, A. D. Myers, M. A. Strauss, K. B. Schmidt,

Ž. Ivezic, N. P. Ross, C. L. MacLeod, and R. Riegel, "Quasar classification using color and variability," *The Astrophysical Journal*, vol. 811, no. 2, p. 95, 2015.

[69] I. Sullivan. (2016) Starfast - a fast simulation building tool for testing algorithms. [Online]. Available: https://dmtn-012.lsst.io/

Vita

Matthias A. Lee received his Bachelor of Science in Computer Science from Wentworth Institute of Technology (Boston, MA) in 2011 and completed his Masters of Engineering in Computer Science from Johns Hopkins University (Baltimore, MD) in 2014. He has spent the past 7 years as a Performance Engineering Co-Op at IBM Rational and IBM Cloudant, developing software performance testing and analysis frameworks. He is also the Technical Lead for the Corrie Health Platform which is aims to reduce hospital readmissions and improve patient outcomes. His research has focused on GPU-acceleration, Image Processing, NoSQL databases, low-power computing and performance testing.